Normative Regulation of Open Multi-agent Systems

Andrés García-Camino

Directors:
Dr. Pablo Noriega Blanco-Vigil
Dr. Juan Antonio Rodríguez Aguilar
i Dr. Wamberto Weber Vasconcelos

Tutor: Dr. Josep Puyol Gruart

Departament de Ciències de la Computació i Intel·ligència Artificial
Escola Tècnica Superior d'Enginyeria
Universitat Autònoma de Barcelona
2008

A mis padres y a mi hermano.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 2
- 1.2 Contributions . . . 4
- 1.3 Structure . . . 6
- 1.4 Publications . . . 7

2 Related Work **11**
- 2.1 Logics of Norms . . . 11
- 2.2 Normative Computational Languages . . . 14
  - 2.2.1 Conditional Deontic Logic with Deadlines . . . 14
  - 2.2.2 Z Specification of Norms . . . 15
  - 2.2.3 Event Calculus . . . 16
  - 2.2.4 Rights and Obligations . . . 17
  - 2.2.5 NoA Agent Architecture . . . 18
  - 2.2.6 Social Integrity Constraints . . . 18
  - 2.2.7 Object Constraint Language . . . 19
  - 2.2.8 Hybrid Metric Interval Temporal Logic . . . 20
- 2.3 Models for Regulated MAS . . . 21
  - 2.3.1 Electronic Institutions . . . 22
  - 2.3.2 MOISE and MOISE$^+$ . . . 24
  - 2.3.3 Commitment-based Institutions . . . 25
  - 2.3.4 OperA . . . 26
  - 2.3.5 LGI Model . . . 27
  - 2.3.6 Electronic Institutions for Virtual Organisations . . . 28
  - 2.3.7 Multi-agent Policy Architecture . . . 28
- 2.4 Conclusions . . . 29

3 Regulating Activities in Electronic Institutions **33**
- 3.1 A Normative Language for Electronic Institutions . . . 34
  - 3.1.1 Examples . . . 35
- 3.2 Executable Norms . . . 37
  - 3.2.1 Jess . . . 37
  - 3.2.2 Norm implementation . . . 38
- 3.3 Developing and Deploying Norms . . . 41

3.3.1 Automatic Translation of Norms . . . 41
3.3.2 Integration with Electronic Institutions . . . 42
3.4 Conclusions . . . 43

4 Constraint-based Regulation **47**
4.1 A Rule Language for Managing Normative Positions . . . 48
4.1.1 Preliminary Definitions . . . 50
4.1.2 A Language for Rules with Constraints . . . 51
4.1.3 Semantics of Rules . . . 52
4.1.4 An Interpreter for Rules with Constraints . . . 54
4.1.5 Pragmatics of Rules with Constraints . . . 55
4.2 Programming Institutional Rules . . . 56
4.2.1 Institutional States . . . 57
4.3 Providing Semantics to Deontic Notions . . . 58
4.4 Normative Conflict Resolution . . . 60
4.4.1 Representing and Enacting Protocols via Institutional Rules 61
4.4.2 Example: The Dutch Auction Protocol . . . 62
4.5 Conclusions . . . 65

5 Regulating Concurrency **67**
5.1 $\mathcal{I}$: A Language for Institutions . . . 68
5.1.1 Semantics . . . 70
5.1.2 Operational Semantics . . . 71
5.1.3 Interpreter . . . 75
5.2 Example of Concurrency: Soup Bowl Lifting . . . 77
5.3 Applied Example: Bank . . . 78
5.4 Norm-Oriented Programming of Scenes . . . 80
5.4.1 Providing Semantics to Normative Positions . . . 80
5.4.2 Normative Conflict Resolution . . . 81
5.5 Conclusions . . . 82

6 A Normative Structure for Multiple Activities **85**
6.1 Scenario . . . 86
6.2 Normative States, Transitions and Structures . . . 87
6.2.1 Example . . . 90
6.3 Formalising Conflict-Freedom . . . 92
6.3.1 Mapping Normative Structures to Coloured Petri Nets . . 93
6.3.2 Properties of Normative Structures . . . 94
6.4 Resolving Normative Conflicts in Run-time . . . 96
6.4.1 Conflict Detection . . . 96
6.4.2 Conflict Resolution . . . 97
6.5 Conclusions . . . 99

7 Computational Model and Distributed Architecture **101**
7.1 Proposed Distributed Architecture 101
7.1.1 Social Layer Protocols 104
7.2 Conclusions 109

8 Conclusions and Future Work **111**
8.1 Conclusions 111
8.2 Future Work 115
8.2.1 Improving Expressiveness 115
8.2.2 Electronic Institutions 117
8.2.3 New applications using norms 118

A UML Diagrams **119**

B An Interpreter for $\mathcal{I}$ **121**

Bibliography **125**

List of Figures

1.1 This thesis as intersection of research fields. 3
1.2 Comparison of the approaches proposed in each chapter 5

2.1 BNF of Norms from [Vázquez-Salceda et al., 2004] 14
2.2 BNF of Norm Conditions 14
2.3 Z Definition of a Norm from [López y López, 2003] 15
2.4 Main Predicates of Event Calculus 16
2.5 Main Fluents from [Artikis et al., 2005] 17
2.6 EI architecture using AMELI. 23

3.1 Conditional obligation with a deadline 35
3.2 Permission in an interval of time 36
3.3 Sanction related to a deadline violation 36
3.4 Example of a Jess unordered fact 37
3.5 Example of a Jess rule 38
3.6 Example of a conditional obligation with a deadline 39
3.7 Translating the condition 40
3.8 Translating the obligation 41
3.9 Translating the deadline 42
3.10 Rule activation for norm OBLIGED($utter(s, w, i)$ BETWEEN t_1, t_2) 42
3.11 Conditional obligation along a time interval 43
3.12 Implementation of a conditional obligation along a time interval . 43
3.13 Developing and deploying norms 44
3.14 AMELI and the JessNormEngine Service 45

4.1 Semantics as a Sequence of Δ's 50
4.2 Interpreter for Rules with Constraints 54
4.3 Sample Institutional State 58
4.4 The Dutch Auction Protocol 63

5.1 Semantics as a Sequence of Δ's 68
5.2 Grammar for $\mathcal{I}$ 69
5.3 The `s_star` predicate 76
5.4 The `s_if` predicate 76
5.5 The `fire` predicate 76

5.6 The `s_force` predicate 77
5.7 The `s_eca` predicate 77

6.1 Activities of Virtual Marketplace 87
6.2 An Interpreter of Normative Transitions 89
6.3 Activities and Normative Structure 91
6.4 Evolution of CPNs for an Activity and a Normative Structure . . 95
6.5 Norm Adoption Algorithm 98

7.1 AMELI$^+$architecture 103
7.2 Communication channels involved in the activation of a rule . . . 104
7.3 Enactment of a normative transition rule 105
7.4 Response of Governor agents to external agents' attempts 106
7.5 Response of a normative manager to a normative command . . . 106
7.6 Response of a scene manager to a forwarded attempt 107
7.7 Response of a scene manager to a normative command 108

8.1 Comparison of proposed languages 116

A.1 Interfaces between AMELI and norm engine of Chapter 3 119
A.2 UML Diagram of the Automatic Translator of Chapter 3 120

List of Tables

1.1 Comparison of the different approaches of chapters 3, 4, and 5 . . 4

2.1 Comparison of norm languages 29
2.2 Comparison of models of regulated MAS 30

8.1 Final comparison of the different norm languages 112
8.2 Final comparison of the different models of regulated MAS . . . 115

Acknowledgements

First of all, my deepest acknowledgements are for my advisors, Pablo Noriega, Juan Antonio Rodríguez, and Wamberto Vasconcelos, I could not have completed this thesis without them.

A mis padres, por sacrificarse tanto, por ayudarme en los momentos duros y no tan duros de este camino.

A los de siempre (Víctor, Jose y Alberto), por estar siempre ahí.

A Eva Bou, Andrea Giovannucci, Eloi Puertas, Raquel Ros y al resto de doctorandos del IIIA, por compartir buenos momentos juntos a lo largo de este camino.

Querría dar las gracias a todo el personal del IIIA, tanto científico como administrativo, por la inestimable ayuda que siempre me han ofrecido.

Thanks to Marek Sergot, Keith Clarke and Robert Craven who hosted me during my short stage in Imperial College London. Their influence undoubtedly is present in $\mathcal{I}$, the language presented in chapter 5 of this thesis.

Thanks to Dorian Gaertner and Martin Kollingbaum to remind me that research is a team work.

Thanks to Tim Norman and Wamberto Vasconcelos who hosted me during my short stage in the University of Aberdeen.

Este trabajo ha sido parcialmente financiado por una beca I3P del CSIC y los proyectos de investigación TIC-2003-08763-C02-00, TIN2006-15662-C02-01 y 2006-5-0I-099.

"Imagination is more important that knowledge. For knowledge is limited, whereas imagination embraces the entire world, stimulating progress, giving birth to evolution."

(Albert Einstein, *"What Life Means to Einstein" in The Saturday Evening Post (26 October 1929)*)

Chapter 1

Introduction

Autonomous agents and multi-agent systems (MAS) are a recent approach to analysing, designing and implementing complex software systems. Using agents as a key abstraction provides a large and powerful collection of metaphors, methodologies and tools that have allowed conceiving and implementing many innovative types of software [Jennings et al., 1998].

One of the open questions posed in [Jennings et al., 1998] about MAS was: "How to avoid or mitigate harmful overall system behaviour, such as chaotic or oscillatory behaviour?". One possible answer to this question is researchers' proposal to regulate MAS by arranging agents in organisations or institutions.

Organisations and institutions are a key metaphor to regulate interactions of self-interested agents because of the above-mentioned properties. For instance, in a virtual market agents are usually classified as providers or customers, and purchasing activities may be classified as different types of auctions with their particular rules. These auctions guide and restrict the behaviour of agents in order to acquire goods. Furthermore, in a virtual market new providers and customers may appear or the existing ones may decide to cease its trading behaviour for a long period of time.

Norms are an intrinsic part of human organisations and institutions. Norms have subdued human societies for centuries thanks mainly to their fear to punishments when violating them. Although agents cannot feel fear, they can emulate it by reasoning for instance about the money loss of the fine(s) when transgressing norms. Initially, agents were regimented following standard software methodologies where software executes a given set of commands in a predetermined order. However, as the agent community agreed autonomy is an essential feature of agents, modelling agent behaviour with norms has been gaining popularity as they allow the partial characterisation of desirable actions of agents.

The purpose of this thesis is to explore how to computationally realise norms in open, regulated, and structured MAS.

1.1 Motivation

In this section, using a motivating scenario, we pose the questions that this thesis tries to answer. The scenario used throughout this thesis is a supply-chain scenario in which companies and individuals come together at a virtual (electronic) marketplace to conduct business.

Consider a wire factory *WireMaking Ltd.* that starts an auction to find suppliers of copper. A buyer agent starts an auction for *WireMaking Ltd.* for copper to which supplier agents may respond. A bid consists of the number of kilograms of the given prime material. Then, after receiving bids the auctioneer assesses the best offer(s).

To discourage unfulfilled promises, buyers may establish economical sanctions or other persuasive measures to be applied in case of delivery problems. Thus, the first setting is the regulation of one activity with norms of behaviour and the following question arises: What kind of language should be used to specify these norms of behaviour?

Some attempts to answer this question have been made, *e.g.* [Esteva, 2003] or [López y López, 2003][1]. However, an essential feature has not been thoroughly treated by these approaches while establishing desirable agent behaviour, *i.e.* time requirements. For instance, the wire factory may want a certain amount of copper to be delivered by the end of the week after the auction takes place. Therefore, after winning an auction, the suppliers are expected to deliver the promised quantity of copper on time. Although the management of time requirements is a desirable feature of the language we are searching for, we notice that dealing with further constraints is also necessary. For instance, when a supplier wins an auction it is expected to deliver the goods by the deadline, but also to fulfil the quantity requirements claimed in the bid. Thus, two questions that this thesis tries to answer are:

Q.1 How to specify norms that regulate a multi-agent activity?

Q.2 How to make norms operational to regulate a multi-agent activity?

Retaking our supply chain example, *WireMaking Ltd.* may settle the bill from time to time for the delivery of the promised quantity of engines. That is, apart from the auction activity, agents may participate in a delivery activity where goods and money are changed according to the agreement established by successfully finishing an auction. Thus, when a supplier wins an auction, it is expected to deliver the promised goods by participating in the delivery activity.

In this setting, actions in one activity may have effect on other activities, *e.g.* a bid in an auction may generate the expectation of copper deliveries. Furthermore, when the MAS is composed of a large number of activities, it is desirable to distribute them among several computers to reduce their workload and provide a smoother and faster evolution of the activities. Thus, the second setting we shall consider is the regulation of multiple distributed activities. From the previous statements, a question that arises is:

[1] See Chapter 2 for a more comprehensive list of work on norm languages.

Q.3 How to specify norms that handle multiple concurrent activities?

In the literature it is admitted that norms may be contradictory [Sartor, 1992]. Let us consider that a norm may allow all suppliers in an auction to bid at a given moment. However, another norm may state that *ShinningCopper Co.*, a specific supplier agent, is forbidden to bid because of previous unfulfilled deadlines on delivery. In [Kollingbaum, 2005], agents are provided with mechanisms to resolve conflicts among norms. However, the norms have to be *conflict-free* in order to apply them. For instance, what should the MAS do? Should the MAS allow the bid or should it punish the agent? Thus, in our opinion, the resolution of conflicts among norms should be applied in the activity by the MAS.

Since our main goal is to regulate with norms MASs with multiple distributed activities, the following questions also arise:

Q.4 How to make norms operational to regulate multiple concurrent and distributed activities?

Q.5 How to computationally enact distributed regulation?

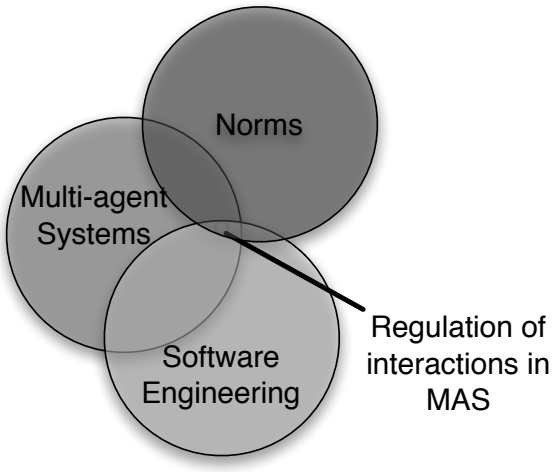

Figure 1.1: This thesis as intersection of research fields.

The questions posed above are the main concerns of this thesis. Figure 1.1 shows where the problem addressed in this thesis are. The main concern is to provide means to build a computational realisation of a class of MASs regulated by norms. To accomplish this task we will benefit from studies on norms like *deontic logics* [von Wright, 1951] and from already built, regulated MAS such as *electronic institutions* [Rodríguez-Aguilar, 2001].

1.2 Contributions

In this section we introduce the key contributions of this thesis. In order to answer our research question about how to specify norms that regulate a multi-agent activity, we provide three norm languages. First, we propose a high-level language that allows the user to specify temporal aspects of norms, *e.g.* activation, deactivation and deadlines. Second, we propose a rule language to specify norms with arithmetic constraints. Third, we propose a language with different types of rules to specify norms (with temporal aspects and arithmetical constraints) over agents' simultaneous speech acts and to specify preventive and corrective actions that the system has to perform in each case.

Features	Approach		
	Jess Norms (Ch. 3)	IRL (Ch. 4)	$\mathcal{I}$ (Ch. 5)
Constraints	time	management	management
Distribution	centralised	one activity	one activity
Concurrent Behaviour	activities	actions	actions
Concurrent Regulation	one action	one action	simultaneous actions
Rule execution	forward-chaining	no forward-chaining	regulative rules

Table 1.1: Comparison of the different approaches of chapters 3, 4, and 5

Table 1.1 shows a comparison of features of the three languages. In order to compare normative languages we use the following features:

Constraints – This feature depicts the degree of constraint management. We distinguish no specification (–), specification only of time constraints (time), specification of constraints (specification) and specification and modification (management).

Distribution – This feature reflects the degree of distribution of norms. We distinguish no distribution of norms (centralised), norms distributed in each agent (agents) and norms distributed in each activity (activities).

Concurrent Behaviour – This feature shows the degree of concurrency on actions. We distringuish no concurrency (–), concurrent actions in one or no activity (actions), and concurrent actions in concurrent activities (activities).

Concurrent Regulation – This feature depicts the degree of regulation on concurrent actions. We distinguish:

- no regulation of actions (–),
- no regulation of actions but regulation of goals (goals),
- just monitoring and sanctioning of actions (monitoring),
- monitoring, sanctioning and prevention of one action at a time (one action),
- monitoring, sanctioning and prevention of simultaneous actions (simultaneous actions).

Rule execution – This feature shows how rules are triggered. We distinguish rules are triggered until no new rule can be triggered (forward-chaining), one execution per rule with different parameters (no forward-chaining) and (regulative rules) different types of rules that change the execution of forward-chaining and one-execution rules.

Jess norms is the first norm language dealing with time in electronic institutions. However, it does not manage arithmetic constraints. IRL is the first language to include constraint management and the specification of the effects of valid events. Notice that it has improved the aspect of constraint management. However, it does not regulate a set of simultaneous actions. Finally, $\mathcal{I}$ is the first language to include that aspect allowing, *e.g.*, to prevent simultaneous actions from being performed. For instance, we may avoid the modification by different agents of the same variable at the same time. As for rule execution, Jess norms uses a standard production system with forward chaining. However, IRL is a rule-based system without forward chaining. Finally, language $\mathcal{I}$ provides several types of rules: standard production rules with forward chaining, standard reactive rules without forward chaining and rules to modify the execution of previous rules, *e.g.* by ignoring agents' actions or by preventing certain states.

In order to answer our research question about how to make norms operational to regulate a multi-agent activity, we provide an implementation for the automatic translation of norms in our high-level language of chapter 3 into rules of a standard production system. Using this automatic translation, we also implemented a norm service for electronic institutions that complements the regulation of activities. Furthermore, we provided an implementation of interpreters for the non-standard rule languages proposed in chapters 4–6.

In order to answer our research question about how to specify norms that handle multiple concurrent activities, we propose a rule language for the propagation of normative positions of agents among activities that we refer to as the language of the normative structure.

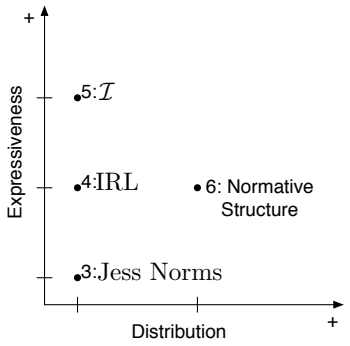

Figure 1.2: Comparison of the approaches proposed in each chapter

Figure 1.2 shows a comparison of the proposed languages in terms of expressiveness and distribution. On the one hand, the languages of chapters 3 - 5 grow in expressiveness without dealing with distribution. On the other hand, the language of chapter 6 deals with distribution of activities in the same degree of expressiveness as that of the language presented in chapter 4.

In order to answer our research question about how to make norms operational to regulate multiple concurrent and distributed activities, we provide normative structures, a computational model for the propagation of normative positions, and an algorithm to manage conflicting norms at runtime to complement formal verification techniques. Our algorithm resolves conflicts among norms at runtime (possibly enriched with arithmetical constraints). If the system does not resolve a given conflict in the normative structure, agents can use conflict resolution techniques to decide which conflicting normative position to comply with. Supplementing formal verification techniques with practical correction techniques used at runtime is not new in software engineering. Usually, imperative programming languages have constructs to detect and repair runtime errors or exceptions. However, these techniques have not been applied before at runtime to norms with constraints. The use of our proposed algorithm enables software designers to correct the specification of desired behaviour of software components at runtime.

Finally, in order to address our research question about how to computationally enact distributed regulation a distributed architecture is presented for the enactment of a regulated MAS. The architecture supports:

- the regulation of agent activities with norms activated as result of agent behaviour,
- the activation of norms among activities and
- the resolution of conflicts among norms in an activity at runtime.

This architecture establishes the basis for building MAS enriched to enforce, propagate, and resolve conflicts in, sets of norms at runtime. Furthermore, by providing this architecture to electronic institutions we enable these to incorporate all the conceptual contributions we have proposed.

1.3 Structure

This thesis is organised as follows:

Chapter 2 surveys the work on the topics dealt with in this thesis. The chapter is divided into an overview on norms in deontic logics and multi-agent systems, on computational normative languages and on regulated multi-agent systems.

Chapter 3 starts addressing **Q.1** (normative language) and **Q.2** (computational model) in a centralised manner and introduces a language for the representation of norms in electronic institutions and its translation into Jess rules to give

them an operational semantics. These norms are complemented with temporal operators to establish deadlines and the time of activation and deactivation.

In chapters 4 and 5, we start by regulating a single scene:

Chapter 4 presents IRL, a language for the representation and explicit management of normative positions, *i.e.* permissions, prohibitions, and obligations active at runtime. This language replaces the language of chapter 3 in the pursuit of **Q.1** (normative language) and **Q.2** (computational model). This language is supplemented with constraints conferring normative positions with more expressiveness since not only temporal constraints can be represented with this language. In this chapter, we introduce the notion of normative positions, differentiate various types of prohibitions and obligations and provide the means to implement activities without taking into account relations among them.

Chapter 5 describes $\mathcal{I}$, a language for the representation of normative positions, *i.e.* permissions, prohibitions, and obligations active at runtime, and the system behaviour given these normative positions. The system, following its specification, ignores, forces, expects events or prevents states of affairs. In this chapter, we establish how to enforce normative positions by providing the specification of the system behaviour. Furthermore, we classify the usual behaviour of systems that enforce normative positions and also we extend it with the notion of preventing a state or ignoring, forcing, expecting or sanctioning sets of simultaneous events. This language replaces the language of chapter 4 in the pursuit of **Q.1** (normative language) and **Q.2** (computational model).

Then, in chapters 6 and 7, we regulate multiple distributed activities:

Chapter 6 addresses **Q.3**, introducing the normative structure, an additional layer in the MAS model that propagates the effects of actions among activities, and **Q.4** by presenting an algorithm for the resolution of normative conflicts. In this chapter, we extend the behaviour of the system with the actions of propagating and resolving conflicts among normative positions, *i.e.* active norms. In addition, we propose that activities specified in different languages can propagate normative positions by means of the normative structure.

Chapter 7 addresses **Q.5** by describing AMELI$^+$, an extension of the agent middleware for electronic institutions that incorporates the enactment of the normative structure propagating formulae, including normative positions, among several activities. These activities are regulated by normative positions and the specification of their behaviour using languages as the ones presented in chapters 4 and 5. The architecture also embeds the algorithm presented in chapter 6 for the resolution of normative conflicts.

Chapter 8 discusses the contributions of this research and how it can be extended in the future.

1.4 Publications

The work in Chapter 3 has been published in:

- [García-Camino et al., 2005a] García-Camino, A., Noriega, P., and Rodríguez-Aguilar, J.-A. Implementing Norms in Electronic Institutions. In *Proceed-*

ings of 4th International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS'05), pages 667–673, Utrecht, The Netherlands.

- [García-Camino et al., 2005b] García-Camino, A., Noriega, P., and Rodríguez-Aguilar, J.-A. Implementing Norms in Electronic Institutions (Extended Abstract). In *3rd European Workshop on Multi-agent Systems (EUMAS'05)*, Brussels, Belgium.

The work in Chapter 4 has been published in:

- [García-Camino et al., 2008] García-Camino, A., Rodríguez-Aguilar, J. A., Sierra, C., and Vasconcelos, W. (2008). Constraint rule-based programming of norms for electronic institutions. *Journal on Autonomous Agents and Multi-Agent Systems*. (In press).

- [García-Camino et al., 2006a] García-Camino, A., Rodríguez-Aguilar, J.-A., Sierra, C., and Vasconcelos, W. A Distributed Architecture for Norm-Aware Agent Societies. In Baldoni, M. et al., editors, *Declarative Agent Languages and Technologies III*, volume 3904 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 89–105. Springer, Berlin Heidelberg.

- [García-Camino et al., 2006b] García-Camino, A., Rodríguez-Aguilar, J.-A., Sierra, C., and Vasconcelos, W. A Rule-based Approach to Norm-Oriented Programming of Electronic Institutions. *ACM SIGecom Exchanges*, 5(5):33–40.

- [García-Camino et al., 2006c] García-Camino, A., Rodríguez-Aguilar, J.-A., Sierra, C., and Vasconcelos, W. Norm Oriented Programming of Electronic Institutions. In *Proceedings of 5th International Joint Conference on Autonomous Agents and Multiagent Systems. (AAMAS'06)*.

- [García-Camino et al., 2007b] García-Camino, A., Rodríguez-Aguilar, J.-A., Sierra, C., and Vasconcelos, W. Norm-Oriented Programming of Electronic Institutions: A Rule-based Approach. In *Coordination, Organization, Institutions and Norms in agent systems II*, volume 4386 of *Lecture Notes in Computer Science*, pages 177–193. Springer-Verlag.

- [García-Camino et al., 2006d] García-Camino, A., Rodríguez-Aguilar, J.-A., Sierra, C., and Vasconcelos, W. Norm-Oriented Programming of Electronic Institutions (Extended Abstract). In *Fourth European Workshop on Multi-Agent Systems (EUMAS'06)*, Lisbon, Portugal.

The work in Chapter 5 has been published in:

- [García-Camino, 2007] García-Camino, A. Ignoring, Forcing and Expecting Concurrent Events in Electronic Institutions. In *COIN III: Coordination, Organization, Institutions and Norms in Agent Systems. Revised Selected Papers from the 2007 Workshop Series*, volume 4870 of *Lecture Notes in Computer Science*, pages 15–26. Springer.

The work in Chapter 6 has been published in:

- [García-Camino et al., 2007a] García-Camino, A., Noriega, P., and Rodríguez-Aguilar, J.-A. (2006) An Algorithm for Conflict Resolution in Regulated Compound Activities. In *Engineering Societies in the Agents World VII*, volume 4457 of *Lecture Notes in Computer Science*, pages 193–208.

- [Gaertner et al., 2007] Gaertner, D., García-Camino, A., Noriega, P., Rodríguez-Aguilar, J.-A., and Vasconcelos, W. Distributed Norm Management in Regulated Multi-agent Systems. In *Proceedings of 6th International Joint Conference on Autonomous Agents and Multiagent Systems. (AAMAS'07).*

- [Gaertner et al., 2008] Gaertner, D., García-Camino, A., Noriega, P., Rodríguez-Aguilar, J. A., and Vasconcelos, W. (2008). Normative structures for regulating open multi-agent systems. *Journal on Autonomous Agents and Multi-Agent Systems.* (submitted).

- [Kollingbaum et al., 2007a] Kollingbaum, M. J., Vasconcelos, W. W., García-Camino, A., and Norman, T. J. Conflict resolution in norm-regulated environments via unification and constraints. In *Declarative Agent Languages and Technologies V*, volume 4897 of *Lecture Notes in Artificial Intelligence*, pages 158–174. Springer.

- [Kollingbaum et al., 2007b] Kollingbaum, M. J., Vasconcelos, W. W., García-Camino, A., and Norman, T. J. Managing conflict resolution in norm-regulated environments. In *Engineering Societies in the Agents World VIII*, volume (In press) of *Lecture Notes in Artificial Intelligence.* Springer.

The work in Chapter 7 has been published in:

- [García-Camino et al., 2007c] García-Camino, A., Rodríguez-Aguilar, J. A., and Vasconcelos, W. (2007). A Distributed Architecture for Norm Management in Multi-Agent Systems. In *COIN III: Coordination, Organization, Institutions and Norms in Agent Systems. Revised Selected Papers from the 2007 Workshop Series*, volume 4870 of *Lecture Notes in Computer Science*, pages 275–286. Springer.

Chapter 2

Related Work

Apart from classical studies on law, research on norms and agents has been addressed by two different disciplines: sociology and philosophy. On the one hand, socially-oriented contributions highlight the importance of norms in agent behaviour (*e.g.*, [Conte and Castelfranchi, 1995b, Conte and Castelfranchi, 1993, Tuomela and Bonnevier-Tuomela, 1995]) or analyse the emergence of norms in multi-agent systems (for instance, [Walker and Wooldridge, 1995] or the work in [Shoham and Tennenholtz, 1995]). On the other hand, logic-oriented contributions focus on the deontic logics required to model normative modalities along with their paradoxes (*e.g.*, [von Wright, 1963] [Alchourron and Bulygin, 1981] [Lomuscio and Nute, 2004]). The last few years, however, have seen significant work on norms in multi-agent systems, and norm formalisation has emerged as an important research topic in the literature (for example, [Fornara et al., 2004] [Dignum, 1999] [Boella and van der Torre, 2003] [Vázquez-Salceda et al., 2004]).

In this chapter, we present relevant related work in the following topics: logics of norms, normative computational languages and Regulated MASs.

2.1 Logics of Norms

Norms have been widely studied in several fields such as logics, sociology, psychology, and law. In this section we focus on the logics that study norms, *i.e.* deontic logics, since the languages on this thesis are inspired on them.

Although prior to Wright philosophers have remarked on the formal logical relations of deontic concepts and compared deontic concepts with *alethic* ones[1] [Huisjes, 1981, Knuuttila, 1981], Wright was the first one to propose a plausible deontic logic in [von Wright, 1951]. It was found later on that a deontic logic could be given a Kripke semantics [Føllesdal and Hilpinen, 1981].

Further on, the work in [Meyer, 1987] reduces deontic logic to a variant of dynamic logic [Harel, 1979] that represents the consequences of prohibited ac-

[1] Alethic concepts are those denoting *modalities* of truth, such as necessity, contingency, or impossibility.

tions in normative systems in terms of violations. Although in our work we do not explicitly manage violations, we incorporate them in Chapters 3, 4 and 5 as attempts that are forbidden. In our architecture, institutions are specified by means of rules establishing how they will react after those violations.

The problem with these normative systems are paradoxes [Åqvist, 1994]. For example, Ross' paradox [Ross, 1941]. Consider the following statements (1 and 2) that we could represent in logic as (3 and 4) using the substitution (m = "the letter is mailed" and b = "the letter is burned"):

1. It is obligatory that the letter is mailed
2. It is obligatory that the letter is mailed or the letter is burned
3. Om
4. $O(m \vee b)$
5. $p \rightarrow (p \vee q)$

RM. $r \rightarrow s \vdash Op \rightarrow Os$

Statement 5 is a tautology[2] in the logic. Rule RM is an inference rule that can be applied whenever the left-hand side holds. Applying RM on 5 we can see that 4 follows 3, but it is odd to say that an obligation to mail the letter entails an obligation that can be fulfilled by burning the letter, something presumably forbidden.

To solve these and other paradoxes, new systems of deontic logic have been proposed. It continues to be a matter of study and workshops on Deontic Logic in computer science have been arranged since 1991 being biannual since 1994 [DEON, 2006].

[Sergot, 2001] proposes to generalise the Kanger-Lindahl theory of normative positions. In the literature the term *normative positions* has been used to refer to duties and privileges between two agents. More concretely, it refers to a point in the space of possible relations between two agents. In his work, Sergot presents normative positions among any agent. An example of normative position extracted from [Sergot, 2001] is :

$$PE_aF \wedge \neg\, PE_b\neg\, F \wedge \neg\, OE_aF \wedge P(E_aF \wedge \neg\, E_bF)$$

Expressing the right (P) of agent a to perform (E_a) the erection a fence between a's and b's houses (F) and meaning that agent a is permitted to see to it that F, agent b is not permitted to hinder the erection of the fence, agent a is not obliged to see to it that F, and that agent a's actions are not dependent on b's actions.

In this thesis we adopt a simpler notion of normative position to refer to each atomic formula in the current state of relations between two agents. These atomic formulae represent whether an agent is permitted, forbidden or obliged

[2] A tautology is a statement which holds for all truth values of its atomic propositions.

to perform an action. For instance, the fact that agent a is obliged to deliver good g to agent b will be regarded as a normative position. We will see different notations for this in chapters 3, 4, and 5.

The work in [Broersen et al., 2004] proposes a deontic logic to represent obligations with deadlines based on CTL (Computational Tree Logic), a temporal logic. They start by defining obligations with deadlines in terms of violations: an obligation regarding formula ρ to hold with deadline δ holds if and only if a violation holds when ρ does not hold before the deadline δ. However, they found a counter-intertuitive property and decided to redefine obligations with deadlines in terms of achievements: an obligation of formula ρ to hold with deadline δ holds if and only if there exists a time-point before the deadline δ where formula ρ holds. The property in question is if there is no possibility of ρ not to hold before the deadline δ then ρ is obligatory before δ. They argue that is counter-intuitive because if ρ is unavoidable, then the obligation to meet the deadline is void, since it concerns an achievement that is already met. They avoid this property with their definition of obligation with deadline in terms of achievement. This kind of obligations are partial, that is, some real applications cannot be implemented with them. For instance, if agents are obliged to not smoke even they violate (or achieve) this obligation they should stiil being obliged to not do it so. Furthermore, as they are not raising violations anymore, obligations just disappear. We find an interesting work of research to redefine obligations with deadline based both in achievement and violations and analyse the properties of the newly created normative system.

In the literature, there are two views of obligations. On the one hand, an obligation is resembled to the alethic notion of necessity, i.e. there is no possibility that the contrary happens. In [Sergot, 2001] this notion of necessity is used in obligatory actions: "It is obligatory that a brings about F". For instance, if I always cross the main door of the building towards the exterior, I am outside the building. If it is obligatory that I cross the main door towards the exterior, then it is obligatory for me to be outside the building. On the other hand, an obligation is the expectation of someone regarding something to be in a certain way or to something to happen, i.e. there are possibilities to not fulfil the expectations. When the expectations about actions are not fulfilled, they are commonly called violations [Conte and Castelfranchi, 1995a]. For instance, in case of fire I am obliged to cross the main door of the building towards the exterior. However, I am free to leave the building through a safer or closer exit if it is appropriate.

Basically, the paradoxes of deontic logic appear since they include connectives, such as the implication, inside the deontic operators. The emergence of these paradoxes are the result of the interpretation of the same obligation operator in both senses (necessity and expectation). Thus, there is a need to differentiate both operators properly.

2.2 Normative Computational Languages

In this section we compare a representative sample of norm languages for MAS.

2.2.1 Conditional Deontic Logic with Deadlines

Vázquez-Salceda *et al.* [Vázquez-Salceda et al., 2004] propose the use of a deontic logic with deadline operators. These operators specify the time or the event after (or before) which a norm is valid. This deontic logic includes obligations, permissions and prohibitions, possibly conditional, over agents' actions or predicates.

As shown in figure 2.1, a norm as defined in [Vázquez-Salceda et al., 2004] is composed of several parts. The norm condition is the declaration of the context

NORM	::=	NORM_CONDITION VIOLATION_CONDITION DETECTION_MECHANISM SANCTIONS REPAIRS
VIOLATION_CONDITION	::=	*formula*
DETECTION_MECHANISM	::=	{*action expressions*}
SANCTIONS	::=	PLAN
REPAIRS	::=	PLAN
PLAN	::=	*action expression* | *action expression* ; PLAN

Figure 2.1: BNF of Norms from [Vázquez-Salceda et al., 2004]

in which the norm applies. The other fields in the norm description are: 1) the *violation condition*, a formula defining when the norm is violated; 2) the *detection mechanism* that describes the mechanisms included in the agent platform that can be used for detecting violations; 3) the *sanctions* defining the actions that are used to punish an agent's violation of the norm; and 4) the *repairs*, a set of actions that are used for recovering the system after the occurrence of a violation. As the definition in figure 2.2 shows, norms can be deontic notions

NORM_CONDITION	::=	$N(S\ \langle \text{IF}\ C\rangle)$ | OBLIGED$(a$ ENFORCE$(N(a, S\langle \text{IF}\ C\rangle)))$
N	::=	OBLIGED | PERMITTED | FORBIDDEN
J	::=	(a, P) | $(a$ DO $A)$
S	::=	J | J TIME | J ACTION
C	::=	*formula*
P	::=	*predicate*
A	::=	*action expression*
TIME	::=	BEFORE D | AFTER D
ACTION	::=	BEFORE J | AFTER J

Figure 2.2: BNF of Norm Conditions

such as permissions, obligations or prohibitions. Furthermore, norms can be related to actions or to predicates (states). The former restrict or allow the

actions that a set of agents can perform, the latter constrain the results of the actions that a set of agents can perform. These results are predicates that can hold or not. For instance, It is forbidden that tom performs the action of smoke (FORBIDDEN (tom DO $smoke$)) and it is forbidden that tom brings about that the air is polluted (FORBIDDEN ($tom, polluted(air)$)) are two examples of the types of norm stated above.

Through the condition (C) and temporal operators (BEFORE and AFTER), norms can be made applicable only to certain situations. Conditions and temporal operators are considered optional. Temporal operators can be applied to a deadline (D) or to an action or predicate (J).

Although this approach syntactically includes all the common deontic notions (obligations, permissions, and prohibitions), it does not explicitly manage normative positions. That is, it allows to specify the activation of a norm but not its termination. For example, the authors do not specify if an obligation should still be active or not after its deadline expires. Nevertheless, we took it as basis for the language presented in chapter 3 in which its implementation by the translation of norms into JESS rules is also provided.

Time management is captured by operators BEFORE and AFTER lending more pragmatism to the language but it lacks complex temporal expressions like 'weekly', 'monthly' and so on.

The enforcement mechanisms proposed are limited to alarm and logging mechanisms which, on their own, are not persuasive enough.

2.2.2 Z Specification of Norms

López y López *et al.* [López y López and Luck, 2004] [López y López, 2003] present a model of normative multi-agent system specified in the Z language. Although this work proposes a framework that covers several topics of normative multi-agent systems we shall focus on its definition of norm. Figure 2.3 shows a norm from [López y López, 2003] composed of several parts. In the

Norm

$$addresses, beneficiaries : \mathbb{P}\, Agent$$
$$normativegoals, rewards, punishments : \mathbb{P}\, Goal$$
$$context, exceptions : \mathbb{P}\, EnvState$$

$$normativegoals \neq \emptyset;\ addresses \neq \emptyset;\ context \neq \emptyset$$
$$context \cap exceptions = \emptyset;\ rewards \cap punishments = \emptyset$$

Figure 2.3: Z Definition of a Norm from [López y López, 2003]

schema, *addressees* stands for the set of agents that have to comply with the norm; *beneficiaries* stands for the set of agents that profit from the compliance of the norm; *normativegoals* stands for the set of goals that ought to be achieved

by addressee agents; *rewards* are received by addressee agents if they satisfy the normative goals; *punishments* are imposed to addressee agent when they do not satisfy the normative goals; *context* specifies the preconditions to apply the norm and *exceptions* when it is not applicable. Notice that a norm must always have addressees, normative goals and a context; *rewards* and *punishments* are disjoint sets, and *context* and *exceptions* too.

Their proposal is quite general since the normative goals of a norm do not have a limiting syntax. They also propose the concept of interlocking norms, *i.e.* norms that deactivate other norms. However, the authors do not provide a clear operational semantics to norms nor an explicit management of time. Furthermore, their model assumes that all participating agents have a homogeneous, predetermined architecture. Thus eliminating heterogeneity, another desired feature, which is the freedom of choice for the architecture of the participating agents. Pragmatism is not the principal feature of the proposal since, as mentioned before, the time management is not explicit and a clear operational semantics is not provided.

2.2.3 Event Calculus

Event calculus is used in [Artikis et al., 2005] for the specification of protocols. Event calculus (cf. Figure 2.4) is a formalism to represent reasoning about actions or events and their effects in a logic programming framework and is

Predicate	**Meaning**
happens(Act, T)	Action Act occurs at time T
initially($F = V$)	The value of fluent F is V at time 0
holdsAt($F = V$, T)	The value of fluent F is V at time T
initiates(Act, $F = V$, T)	The occurrence of action Act at time T initiates a period of time for which the value of fluent F is V
terminates(Act, $F = V$, T)	The occurrence of action Act at time T terminates a period of time for which the value of fluent F is V

Figure 2.4: Main Predicates of Event Calculus

based on a many-sorted first-order predicate calculus. Predicates that change with time are called *fluents*. Figure 2.5 shows how obligations, permissions, empowerments, capabilities and sanctions are formalised by means of fluents – prohibitions are not formalised in [Artikis et al., 2005] as a fluent since they assume that every action not permitted is forbidden by default.

Although the initiation and termination of normative positions can be specified with event calculus, [Artikis et al., 2005] does not give the possibility to permit actions by default and only explicitly forbid certain actions. The most notable feature of event calculus is the clarity of its semantics since it is based

Fluent	Domain	Meaning
$requested(S, T)$	boolean	subject S requested the floor at time T
$status$	$\{free,$ $granted(S, T)\}$	the status of the floor: $status = free$ denotes that the floor is free whereas $status = granted(S, T)$ denotes that the floor is granted to subject S until time T
$best_candidate$	agents	the best candidate for the floor
$can(Ag, Act)$	boolean	agent Ag is capable of performing Act
$pow(Ag, Act)$	boolean	agent Ag is empowered to perform Act
$per(Ag, Act)$	boolean	agent Ag is permitted to perform Act
$obl(Ag, Act)$	boolean	agent Ag is obliged to perform Act
$sanction(Ag)$	$\mathbb{Z}^*$	the sanctions of agent Ag

Figure 2.5: Main Fluents from [Artikis et al., 2005]

in logic programming.

Although event calculus models change with time, it lacks on explicit time management functionalities as shown in [Artikis et al., 2005]. Deadlines thus need to be modelled as new events.

Another desired feature is that agents can reason about norms without the need of new complicated machinery. Their deontic fluents are not enough to inform an agent about all types of duties. For instance, to inform an agent that it is obliged to perform an action before a deadline, it is necessary to show the agent the obligation fluent and the part of the theory that models the violation of the deadline.

The enforcement mechanism provided simply amounts to a violation counter. It does not distinguish between violations although some of them are more serious than others. Furthermore, the number of violations an agent has performed is not very persuasive if any of their goals are not hindered, e.g. earn money.

In [Stratulat et al., 2001] (previous to [Artikis et al., 2005]), event calculus is also used to model obligations, permissions, prohibitions and violations.

2.2.4 Rights and Obligations

Michael *et al.* [Michael et al., 2004] propose a formal scripting language to model the essential semantics, namely, rights and obligations, of market mechanisms. Whereas $right(\sharp action, \sharp condition)$ denotes the right to execute action $\sharp action$ whenever condition $\sharp condition$ is true, $obligation(\sharp satisfy, \sharp violate, \sharp punishment)$, denotes the obligation to ensure that condition $\sharp satisfy$ is satisfied no later than condition $\sharp violate$ under the penalty of executing action $\sharp punishment$. For instance, the following obligation states that Bob must ensure that her bank balance goes above 1000 dollars at some time before the end of 2004, under the penalty of 100 dollars being deducted from her account.

$$obligation(balance(bob) > 1000, Time > 12/31/2004, deduct_account(bob, 100))$$

They also formalise a theory to create, destroy and modify objects that either belong to someone or can be shared by others.

Their proposal is suitable to model and implement market mechanisms, and provide an explicit management of temporal constraints. However, they do not provide the full set of normative positions and their language is not pragmatic enough since it uses a complex syntax to denote action effects. Whereas Prolog, the implementing language, has a clear semantics, their work does not completely clarify the operational semantics of the predicates they introduce in their examples.

2.2.5 NoA Agent Architecture

Kollingbaum [Kollingbaum, 2005] proposes a language for the specification of normative concepts (*i.e.*, obligations, prohibitions and permissions) and a programming language for norm-governed reasoning agents. The normative concepts and the programming language are given their operational semantics via the NoA Agent Architecture using Java to explain the meaning of each construct [Kollingbaum and Norman, 2003a] [Kollingbaum and Norman, 2003b].

```
obligation (
      BlockMover,
      achieve clear (b),
      not clear (b),
      clear (b)
)
```

The example above shows how an obligation is represented in NoA. The activity specification within this obligation requires the agent BlockMover to achieve a specific effect, namely "`clear (b)`". To fulfill its obligation, the agent has to choose a plan that produces this effect. Furthermore, the obligation contains the activation condition "`not clear (b)`" and the expiration condition "`clear (b)`".

This approach addresses practical reasoning agents developed using their language, which lacks richer notions such as (temporal) constraints, as well as an architecture – although the approach is practical and has clear advantages such as the possibility to check for norm conflicts and consistency, heterogeneous agents cannot be accommodated. Furthermore, there is no indication of how the proposal adapts to a multi-agent or distributed scenario, as only individual agents are addressed.

2.2.6 Social Integrity Constraints

In [Alberti et al., 2003] the language Social Integrity Constraints (SIC) is proposed. This language's constructs check whether some events have occurred and whether some conditions hold to add new expectations, optionally with constraints.

The syntax of SIC is the following:

$$
\begin{aligned}
\mathit{SIC} &::= [\mathit{ic}_S]^* \\
\mathit{ic}_S &::= \chi \rightarrow \phi \\
\chi &::= \mathit{ExtendedLiteral}\ [\ \wedge \mathit{ExtendedLiteral}\ ]^* \\
\phi &::= \mathit{ExpectList}\ [:\ \mathit{ConstraintList}] \\
\mathit{ExtendedLiteral} &::= \mathit{Expectation}\ \mid\ \mathit{Event}\ \mid\ \mathit{Atom} \\
\mathit{Expectation} &::= \mathbf{E}(\mathit{Term})\ \mid\ \mathbf{NE}(\mathit{Term}) \\
\mathit{Event} &::= \mathbf{H}(\mathit{Term}) \\
\mathit{ExpectList} &::= \mathit{Expectation}\ [\ \wedge\ \mathit{Expectation}\ ]^*
\end{aligned}
$$

Atoms are defined by ground facts in the *Social Organization Knowledge Base*. *ConstraintList* is a conjunction of constraints from Constraint Logic Programming (CLP) [Jaffar et al., 1998].

An example of a SIC construct is:

$$
\left(\begin{array}{l}
\mathbf{H}(\mathit{request}(B, A, P, D, T_r))\wedge \\
\mathbf{H}(\mathit{accept}(B, A, P, D, T_a))\wedge \\
T_r < T_a
\end{array}\right) \rightarrow \mathbf{E}(\mathit{do}(A, B, P, D, T_d)) : T_d < T_a + \tau
$$

The construct above intuitively means "if agent B sent a request P to agent A at time T_r in the context of dialogue D, and A sent an *accept* to B's request at a later time T_a, then A is expected to do P before a deadline $T_a + \tau$".

This proposal does not directly manage normative positions and, thus, their expectations have to be mapped into deontic notions [Alberti et al., 2005]. Since it is implemented using Constraint Handling Rules (CHR) [3], the language is provided with a clear semantics and an explicit management of time as constraints. This is a pragmatic approach in the sense that it is easy to specify norms with temporal constraints. However, the authors do not provide the language with any norm enforcement mechanism since they do not manage obligations but expectations.

2.2.7 Object Constraint Language

Fornara *et al.* in [Fornara et al., 2004], contemporary work with the one presented in chapter 3, propose the use of norms partially written in OCL, the Object Constraint Language [OMG, 2003] which is part of UML (Unified Modelling Language) [OMG, 2005]. Their commitments are used to represent all normative modalities – of special interest is how they deal with permissions: they stand for the absence of commitments. This feature may jeopardise the safety of the system since it is less risky to only permit a set of safe actions, thus forbidding other actions by default. Although this feature can reduce the amount of permitted actions, it allows new or unexpected, risky actions to be carried out.

[3] [Frühwirth and Abdennadher, 2003]

The expression below shows an example of norm written in OCL:

$$
\begin{array}{l}
\mathsf{within}\ h : \mathit{AuctionHouse} \\
\mathsf{on}\ e : \mathit{InstitutionalRelationChange}(h.\mathit{dutchAuction}, \\
\qquad \mathit{auctioneer}, \mathit{created}) \\
\mathsf{if}\ \mathit{true}\ \mathsf{then} \\
\mathsf{foreach}\ \mathit{agent}\ \mathsf{in}\ h.\mathit{employee} \rightarrow \\
\qquad \mathit{select}(\mathit{em} \mid e.\mathit{involved} \rightarrow \mathit{contains}(\mathit{em})) \\
\ \mathsf{do}\ \texttt{makePendingComm}(\mathit{agent}, \\
\qquad \mathit{DutchInstAgent}(\mathit{notSetCurPrice}(\\
\qquad h.\mathit{dutchAuction.id}, \\
\qquad ?p[?p < h.\mathit{agreement.reservationPrice}]), \\
\qquad < \mathit{now}, \mathit{now} + \mathit{time_of}(e1 : \mathit{InstStateChange}(\\
\qquad h.\mathit{dutchAuction}, \mathit{OpenDA}, \mathit{ClosedDA})) >, \forall))
\end{array}
\tag{2.1}
$$

This norm commits the auctioneer not to declare a price lower than the agreed reservation price.

Since Fornara *et al.* adopt OCL, the syntax is a well-defined standard. However, the authors do not mention neither any tool that they are using or plan to use to implement institutions, nor the operational semantics it should have. Time is also managed as constraints but its pragmatism is penalised by the complex syntax of the language. The authors do not provide the language with an explicit management of normative positions since they only consider commitments that can be seen as obligation in deontic logic (Eq. 2.2). Permissions are considered as the presence of commitments (Eq. 2.3) and prohibitions as their absence (Eq. 2.4).

$$\mathsf{C}\phi \leftrightarrow \mathsf{O}\phi \tag{2.2}$$

$$\mathsf{C}\phi \rightarrow \mathsf{P}\phi \tag{2.3}$$

$$\neg\, \mathsf{C}\phi \rightarrow \mathsf{F}\phi \tag{2.4}$$

2.2.8 Hybrid Metric Interval Temporal Logic

In [Cranefield, 2005], contemporary work to our own work in chapter 4, we find a proposal to represent norms via rules written in a modal logic with temporal operators called hyMITL$^{\pm}$. It combines CTL$^{\pm}$ with Metric Interval Temporal Logic (MITL) as well as features of hybrid logics. That proposal uses the technique of formula progression from the TLPlan planning system to monitor social expectations until they are fulfilled or violated.

Formulae of hyMITL$^{\pm}$ are defined via the following grammar:

$$
\begin{array}{rcl}
\phi & ::= & p \mid \neg\,\phi \mid \phi \wedge \phi \mid \forall\, x.\phi_x \mid \mathsf{X}^{+}\phi \mid \mathsf{X}^{-}\phi \\
 & & \phi\, \mathsf{U}_I^{+} \phi \mid \phi\, \mathsf{U}_I^{-} \phi \mid \mathsf{A}\phi \mid \mathsf{E}\phi \mid \downarrow^{u} x.\phi_x \mid I \\
I & ::= & (+\infty, -\infty) \mid [b, b] \mid [b, b) \mid (b, b] \mid (b, b) \\
b & ::= & a \mid +d \mid -d
\end{array}
$$

where

- p is an atomic formula for a first order language L.
- ϕ_x denotes a formula ϕ in which variable x is free (i.e. not bound by $\forall$ or $\downarrow$).
- u is a unit selector on the $\downarrow$ binding operator, referring to the desired granularity of time (e.g. *year* or minute) for binding x to the current time. A value of now indicates maximum precision.
- a and d are terms, possibly containing variables, that denote (respectively) absolute points in time and durations.

In this logic, the temporal operators X (the next/previous state) and U (until) can be applied in the future direction (when adorned with a superscript '+') or the past direction (indicated by a '-'). The meaning of $\mathsf{X}^+\phi$ is that ϕ is true in the next state, and $\phi\,\mathsf{U}_I^+\psi$ assert that ϕ will remain true from the current state for some (possibly empty) sequence of consecutive future states, followed by a state that is within the interval I and for which ψ holds. X^- and U^- are defined similarly, but in the past direction.

A and E are path quantifiers. They assert that the formula that follows the operator applies to all, or respectively at least one, of the possible sequences of states passing through the current state.

[Blackburn et al., 2002] introduces the $\downarrow$ operator, the "binder" operator used in hybrid logics. It binds a variable to a term denoting the current date/time. The optional unit selector u is a time unit constant that indicates that the variable should be bound to the time point resulting from rounding down the current date/time to a particular degree of precision, e.g. to the start of the current year, month or day.

Like in the SIC approach, only expectations are monitored, thus no making an explicit management of normative positions, and no enforcement mechanism is provided. Although the semantics of the language and formula progression tool is clear, it is not so clear how it can be integrated into a MAS thus decreasing its pragmatism. With this approach, the specification of the expected MAS state in the future is possible but the author does not mention how the actual MAS state is accessed, compared with the expected one or the corrective measures to be applied are decided.

2.3 Models for Regulated MAS

In this section we survey work on the specification of co-operative behaviour in MAS.

2.3.1 Electronic Institutions

We start by introducing the reader to one of the first models for regulated MAS not based in regimentation, *i.e.* Electronic Institutions (EIs). [Noriega, 1997] sets the foundations of EIs by proposing dialogical institutions as a means to regulate and implement agent-mediated auctions using a fish market metaphor. These dialogical institutions, subsequently called EIs, were conceived as three components: dialogical framework, performative structure and rules of behaviour.

The *dialogical framework* captured the notion of context by making explicit the participants, their roles and their relevant social interrelationships as well as the communication and the language used in it.

The *performative structure* is a set of interdependent located scenes. Each scene is defined as a set of agents assumed to enact a role by interchanging statements by illocution. These statements are therefore called *illocutions*. Furthermore, this interchange of statements is subject to a protocol, defined by means of a finite-state machine (FSM) where state transitions are labelled by illocutions and the states have associated pending commitments.

Illocutions start with an *illocutionary particle* that declares the intention of the message. For instance, a message informing about the selling of a given good would use the illocutionary particle *inform*.

Finally, as protocols may not be sufficient to make fully explicit how the agent should behave, the *rules of behaviour* complete the specification of the behaviour of each agent. The notion of *governor*, agents in charge of ensuring that institutional communication follows the aforementioned protocols by neglecting forbidden illocutions, is also introduced in [Noriega, 1997].

[Rodríguez-Aguilar, 2001] continues this line of work by proposing a new formalisation of EI which includes a specification language and a computational model. In that work a performative structure specifies the conditions and the order in which agents may enter and leave scenes. Performative structures represent:

- *causal dependencies* (*e.g.* a good cannot be auctioned if it has not been previously registered in the auction house);
- *synchronisation* while entering or leaving a scene;
- *parallelism* by executing several instances of a scene (*e.g.* concurrent auctioning of several items);
- *choice points* that allow roles to select what scenes to enter and leave, and
- *role flow policy* among scenes, *i.e.* what paths can be followed by roles leaving a scene and what scenes they can reach.

Furthermore, this work introduces the need of *normative rules* in order to account the obligations of agents. However, only examples of normative rules are given.

[Esteva, 2003] further extends the work on EIs by improving the formalisation and developing general-purpose software tools to design and interpret institutional specifications to be followed when executing a open set of dialogical agents.

As a result of this thesis, the *Electronic institution development environment* (EIDE) started to be developed. Initially, it only comprised of ISLANDER, an editor and off-line verifier of EI specifications. and AMELI, an agent-based middleware to execute EIs [Esteva et al., 2004]. Currently it also provides aBuilder, an editor that partially completes the development of agents by taking advantage from recurrent particularities of a given specification, and SIMDEI, an online verifier and simulator of specifications.

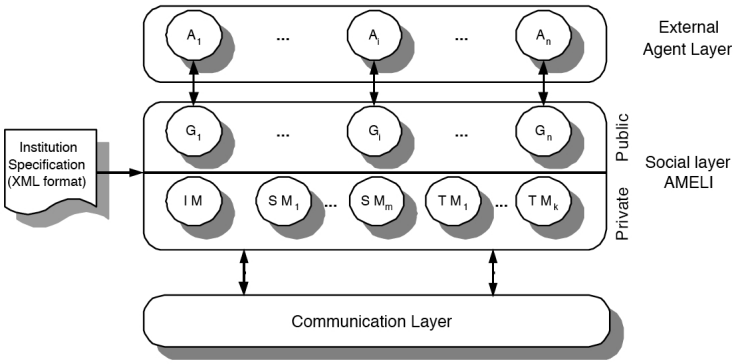

Figure 2.6: EI architecture using AMELI.

[Esteva, 2003] and [Esteva et al., 2004] propose the use of Jess engines to implement norms in EIs. The former proposes one global rule engine per institution, the latter proposes one local rule engine per governor (one per agent). However, none of the previous proposals are optimal since in the former one the rule engine may become a bottleneck in a large MAS and the latter one fails to process information out of the reach of each governor. For instance, when two agents are forbidden to perform certain simultaneous actions, the governor cannot access actions of other agents. In chapter 7 we propose a distributed architecture with rule engines at several levels.

We now focus on the architecture for executing EIs that uses AMELI and is depicted in figure 2.6. It is composed of three layers: 1) external agent layer: external agents taking part in the institution; 2) social layer (AMELI): implementation of the control functionality of the institution infrastructure; and 3) communication layer: in charge of providing a reliable and orderly transport service.

The functionality of AMELI includes: 1) the mediation of speech acts uttered by agents; 2) the provision of key information to successfully participate in the institution; and 3) the institutional enforcement.

The institutional enforcement consists on: 1) guaranteeing the correct evolution of each interaction by filtering erroneous illocutions; 2) guaranteeing that agents movements between scene executions comply with the specification; and 3) controlling which obligations participating agents acquire and fulfil.

The implementation of AMELI presented in [Esteva et al., 2004] realises the above-mentioned functionalities and it is composed of four types of agents:

- **Institution Manager (IM).** It is in charge of starting an EI, authorising agents to enter the institution, as well as managing the creation of new scene executions. It keeps information about all participants and all scene executions. There is one institution manager per institution execution.
- **Transition Manager (TM).** It is in charge of managing a transition controlling agents movements to scenes. There is one transition manager per transition.
- **Scene manager (SM).** Responsible for governing a scene execution (one scene manager per scene execution).
- **Governor (G).** Each one is devoted to mediating the participation of an external agent within the institution. There is one governor per participating agent.

Two of the main assumptions of EIs are that external agents can only communicate with their governors and internals of agents are not accessible by the institution. Therefore, AMELI is independent from any agent architecture allowing heterogeneous agents (including humans) to interact via the EI. However, there is a set of *external* attributes that the EI can access and modify, for example, agents' social status (role, reputation, possessions, etc). Agents' possessions are resources of the environment either real or virtual (money, goods, disk space, etc.).

2.3.2 MOISE and MOISE$^+$

[Hannoun et al., 2000] proposes MOISE, a model to specify organisations of agents based on three major concepts: roles, organisational links and groups. As in other models, *roles* constrain the actions each agent is allowed to perform in each activity. Agents are specified in *groups* that constrain the set of agents they can cooperate with. Organisational links constrain the kinds of interaction that agents may have in the system: communication (regulated by protocols), authority (*e.g.* delegation) or acquaintance (subset of the organisation an agent may use in its reasoning).

They define *organisational structure* as a set of roles, groups and links specifying the structure of the MAS independently from agents; and *organisational entity* as the set of agents functioning under an organisational structure. They associate *missions* to roles. Each mission $m_i = \langle s, G_i, P_i, A_i, R_i \rangle$ specifies an allowed behaviour in the system as a consistent set of goals to achieve (G_i) ,

plans (oriented graphs of actions and sub-goals) to follow (P_i), actions to execute (A_i) and resources to use (R_i). Thus, each element that does not belong to any associated mission is not allowed.

They may equal these sets to $\emptyset$ (nothing is allowed) and Any (all is allowed). The s parameter may be an obligation O (the agent playing the role has no choice, it has to execute the mission) or a permission P (the agent playing the role may decide to execute this mission or not). We notice that, contrary to obligations in EIs that may not be fulfilled, this kind of obligation is imposed in agent's code.

[Hübner et al., 2005] presents MOISE$^+$, an extension of MOISE model allowing agents to not comply with obligations, and S-MOISE, a middleware using MOISE$^+$ to deploy organised MAS.

2.3.3 Commitment-based Institutions

[Colombetti et al., 2002] proposes a commitment-based semantics for agent communication and sketches a model of institution. In their work, a conditional commitment its characterised by the following components:

Debtor. The agent that has the commitment.

Creditor. The agent towards which the commitment is made.

Condition. A state of affairs that "activates" the commitment if it becomes true within a given timeout.

Content. A state of affairs that the debtor is committed to the creditor. It may have an associated *deadline* to become true.

State. A commitment can be in one of six possible states: *unset*, *cancelled*, *pending*, *active*, *fulfilled*, and *violated*.

Thus, they represent a conditional commitment with state s, debtor x, creditor y, condition ϕ, and content φ as the statement $C(s, x, y, \phi, \varphi)$.

They envisage an institution made up of four components: a set of *registration rules*, a set of *interaction rules*, a set of *authorisations*, and an *internal ontology*. They propose registration and interaction rules to be implemented by a collection of commitment-based *protocols*, and authorisations to be implemented as a library of role-dependent actions that can be performed by the members of the group.

In their "Core Institution", they provide three institutional rules that sets the basis for authorisations:

Rule CI-1: any agent is authorised to create an unset commitment with arbitrary debtor, creditor, condition, and content.

Rule CI-2: the debtor of an unset commitment is authorised to set it to either *cancelled* or *pending*.

Rule CI-3 : the creditor of an unset, pending or active commitment is authorised to set it to *cancelled*.

Furthermore, they provide an example of their commitment-based semantics that we partially present here:

- Performing an act of informing amounts to making the unconditional commitment that the content ϕ of the inform act is true, and setting it to pending.
- A request from x to y perform action α if a given condition ϕ holds amounts to create an unset commitment from y to x that y will perform α.
- Accepting a request, *i.e.* an unset commitment, amounts to setting its state to *pending*.
- Refusing a request amounts to setting its state to *cancelled*.

As the authors notice, their definition of institution has been mainly influenced by Searle's analysis of the "count as" relationship [Searle, 1995] and by Jones and Sergot's model of institutionalised power [Jones and Sergot, 1996].

We notice that commitments are always directed towards another agent. Thus, they lack of the concept of "anonymous" commitment. In this case, certain agent becomes a creditor when it creates an obligation that a given agent is committed to do an action but the debtor should not know who his creditor is. To accomplish this using this semantics, we should use a virtual entity towards whom the agent is committed to: the institution. Another instance of this case appears when a role may create commitments but the debtor should not know who is enacting this role, *e.g.* in blind reviews. In this case, the creditor would be a role not a particular agent.

Thus, when using an institution as mediator, we argue that the creditor component is not essential to specify commitments. In this setting, when an agent declares to another agent that it will perform an action involving a third agent (*e.g.* Alice tells Bob that she will talk to Claire), in fact they are making a commitment towards the institution and if its a permitted utterance, it will be accepted and forwarded to its recipient.

2.3.4 OperA

[Dignum, 2003] proposes OperA, a model for specifying organisations in MAS. The three components of an OperA model are:

- **Organisational model:** describes the organisational structure of the society, consisting of roles and their interactions.
- **Social model:** specifies, in terms of agreements, the enactment of roles by individual agents.
- **Interaction model:** describes the possible interaction between agents.

From [Dignum, 2003] we quote the following:

> The *organisational model (OM)* specifies the organizational characteristics of an agent society in terms of four structures: social, interaction, normative and communicative. The *social structure (SS)* specifies objectives of the society, its roles and what kind of model governs coordination. The *interaction structure (IS)* describes interaction moments, as scene scripts, representing a society task that requires the coordinated action of several roles, and gives partial ordering of scene scripts, which specify the intended interactions between roles. Society norms and regulations are specified in the *normative structure (NS)*, expressed in terms of roles and interaction norms. Finally, the *communicative structure (CS)*, specifies the ontologies for description of domain concepts and communication illocutions.

We notice the similarities of OperA with the EI model. The goal of an interaction structure of OperA model is the same as the one of a performative structure. The goals of communicative structure and social structure are captured with a dialogical framework. The normative structure of OperA model is very similar to the notion of norms in EIs since it can be summarised as a set of norms written in LCR. *"LCR is an extension of CTL*, which in turn is an extension of classical propositional logic."*. CTL* is a temporal modal logic whose model of time is a tree-like structure in which the future is not determined [Emerson and Halpern, 1986].

2.3.5 LGI Model

[Minsky, 2005] proposes law-governed interaction (LGI), a decentralised coordination and control mechanism for distributed systems. This middleware allows a possible large, heterogeneous and open set of actors to interact governed by a given policy, called the *interaction law*. The interaction law may be written in Java or Prolog.

In order to enforce the interaction law, a component called *controller* is associated to each actor. The controller is entrusted to mediate the interaction of its actor with others, it decides by interpreting the active law, how to react to messages sent and received by its actor.

Although LGI does not proposes a social model, we notice the similarities between EI governors and LGI controllers. Both realise a local regulation of messages at agent level. However, some applications require regulations that access the state of several agents. To implement this with controllers or governors would require to duplicate in each controller (or governor) the state of all agents possibly involved in a non-local regulation. However, in large systems this is not a viable solution.

2.3.6 Electronic Institutions for Virtual Organisations

[Lopes Cardoso and Oliveira, 2005] proposes a model of EI based on contracts and its use for regulate commerce among Virtual Enterprises. This work is continued in [Lopes Cardoso and Oliveira, 2007] proposing EIs to be defined by *constitutive* and *institutional rules* following the tradition of [Searle, 1995].

Constitutive rules are of the form:

$\langle ConstitutiveRule \rangle ::= \langle BruteFacts \rangle \{ \text{“}\wedge\text{”} \langle AgentRole \rangle \} \text{“}\rightarrow\text{”} \langle InstitutionalFacts \rangle$

Institutional rules work not only on illocutions but also on obligations, their fulfillment and their violations. By executing institutional rules some *institutional procedures* may also be executed. Institutional rules are of the form:

$\langle InstitutionalRule \rangle ::= [\text{“}[\text{”} \langle Context \rangle \text{“}]\text{”}] \langle Antecedent \rangle \text{“}\rightarrow\text{”} \langle Consequent \rangle$

Contemporary to this language, [García-Camino, 2007] proposes $\mathcal{I}$. Although it will we presented in chapter 5, we may remark now that both have certain similar concepts with different names. $\mathcal{I}$ propose *event-condition-action rules* as constitutive rules and *if-rules* as institutional rules. Furthermore, it also proposes three types of rules to *ignore* and *force* concurrent events and *prevent* state of affairs.

2.3.7 Multi-agent Policy Architecture

In [Udupi and Singh, 2006] the authors propose a multi-agent architecture for policy monitoring, compliance checking and enforcement in virtual organisations (VOs). Compared to the distributed architecture that will be presented in chapter 7, their approach also uses a notion of hierarchical enforcement, i.e. the parent assimilates summarised event streams from multiple agents and may initiate further action on the subordinate agents. Depending on its policies, a parent can override the functioning of its children by changing their policies. Instead of considering any notion similar to an EI scene (multi-agent protocol where the number of participants may vary) and assigning an agent exclusively dedicated to the management of one scene, they assign another participant in the VO as parent of a set of agents. Although the parent would receive only the events it needs to monitor, it may receive them from *all* the interactions their children are engaging in. This can be a disadvantage when the number of interactions is large turning the parents into bottlenecks. Although they mention that conflict resolution may be accomplished with their architecture, they leave this feature to the VO agent thus centralising the conflict resolution in each VO. This can also be a disadvantage when the number of interactions is large since the VO agent has to resolve all the possible conflicts. This would require either all the events flowing through the VO agent or the VO agent monitoring the state of the whole VO in order to detect and resolve conflicts. The main theoretical restriction in their approach is that all the agents involved in a change in a policy must share a common parent in the hierarchy of the VO. In an e-commerce example, when a buyer accepts a deal an obligation to supply the purchased item should be added to the seller. However, as they are different

parties, their only common parent is the VO agent converting the latter in a bottleneck in large e-commerce scenarios.

2.4 Conclusions

In this section we compare some features of normative languages and models for regulated MAS.

In order to compare normative languages we use the following features:

Constraints – This feature depicts the degree of constraint management. We distinguish no specification (–), specification only of time constraints (time), specification of constraints (specification) and specification and modification (management).

Distribution – This feature reflects the degree of distribution of norms. We distinguish no distribution of norms (centralised), norms distributed in each agent (agents), and norms distributed in each activity (activities).

Concurrent Behaviour – This feature shows the degree of concurrency on actions. We distringuish no concurrency (–), concurrent actions in one or no activity (actions), and concurrent actions in concurrent activities (activities).

Concurrent Regulation – This feature depicts the degree of regulation on concurrent actions. We distinguish:

- no regulation of actions (–),
- no regulation of actions but regulation of goals (goals),
- just monitoring and sanctioning of actions (monitoring),
- monitoring, sanctioning and prevention of one action at a time (one action),
- monitoring, sanctioning and prevention of simultaneous actions (simultaneous actions).

Language		Features			
		Constraints	Distribution	Conc. Behaviour	Conc. Regulation
1.	CDeonticL	time	centralised	actions	monitoring
2.	Z	–	agents	actions	goals
3.	EC	time	centralised	actions	one action
4.	Rights	specification	centralised	actions	one action
5.	NoA	–	agents	actions	goals
6.	SIC	specification	centralised	actions	monitoring
7.	OCL	specification	centralised	activities	one action
8.	hyMTL	time	centralised	actions	monitoring

Table 2.1: Comparison of norm languages

Observing Table 2.1 we did not find a normative language with the following features: management of constraints, distribution on activities, concurrent activities, and regulation of simultaneous actions. In order to obtain a distributed architecture with these features, we proposed the languages of chapters 3, 4, and 5.

However, to compare models for regulated MAS we use the following features:

Openness – This feature depicts whether new heterogeneous agents may join and leave at runtime. We distinguish closed, closed and heterogeneous and open (and heterogeneous).

Regulation – This feature reflects the degree of regulation. We distinguish: specification of protocols (protocols), specification of protocols and actions agents are expected to perform (protocols and obligations, or protocols and commitments), specification of protocols, obligations, permissions and prohibitions (protocols and norms) and specification of just norms or policies (norms).

Social Structure – We distinguish no social structure (–), just role labels (roles), and role hierarchy.

Activity Structure – This feature depicts the degree of structure of actions. We distinguish: no structure, just a set of actions (actions), different sets of norms apply to just a set of actions (contexts), and a separation of sets of actions (and norms) depending on its purpose (activities).

Models		Features			
		Openness	Regulation	Social Structure	Activity Structure
1.	EIs	open and heterogeneous	protocols and obligations	role hierarchy	activities
2.	MOISE$^+$	open and heterogeneous	protocols	role hierarchy	activities
3.	Commit. Insts.	open and heterogeneous	protocols and commitments	role hierarchy	activities
4.	OperA	open and heterogeneous	protocols and norms	role hierarchy	activities
5.	LGI	open and heterogeneous	norms	roles	actions
6.	EIs for VOs	open and heterogeneous	norms	roles	contexts
7.	MA Policy Arch.	open and heterogeneous	norms	roles	actions

Table 2.2: Comparison of models of regulated MAS

In chapters 3-6 we propose different languages for the regulation of activities. However, we need a computational model to provide our languages a computational realisation. Observing Table 2.2 we obtain desirable features for candidate models of MAS. We envisage an open model of MAS where heterogeneous agents

interact. We prefer these interactions to be regulated only by norms since regulating by protocols and norms requires the MAS programmers to learn how to specify protocols in addition to norms. Furthermore, role hierarchies provide more expressiveness when specifying activities and norms for roles since the specification applies to a role and its subsumed ones. For instance, some activities may be performed or some actions may be forbidden to be bring about by a given set of roles just specifying the subsuming role. Although we plan to include role hierarchies, we acknowledge that the proposal presented in this thesis only deals with role labels. Finally, we envisage a MAS that structures actions and norms in activities allowing a modular design and distribution of the MAS according to the purpose of interactions.

Chapter 3

Regulating Activities in Electronic Institutions

In this chapter we propose a high level language that allows the user to specify temporal aspects of norms, we provide an implementation that performs an automated translation of that high level language into rules of a standard production system, and we implement a norm service for electronic institutions that complements the regulation of scenes.

In [Esteva, 2003], a definition of norms for electronic institutions was provided. This definition was reduced to obligations as a consequence of agents' dialogical actions. However, this definition of norm does not explicitly manage time restrictions. This feature is important since it allows us to express a broader class of norms. That is, the possible activation or deactivation of obligations (or other deontic modalities such as permissions and prohibitions) due to the passing of time. Another interesting application of norms with time restrictions is to specify deadlines for actions. For instance, when an agent wins an auction, it may have 15 minutes to pay for the good. Otherwise, it may be sanctioned and the good re-auctioned. Thus, in this chapter we will redefine norms for electronic institutions and we will give them an operational semantics by means of its translation into standard production rules.

In this chapter we will approach our first two research questions, *i.e.* how to specify norms that regulate a multi-agent activity and how to make them operational. To address the first question, in this chapter we will define a language to be applied in electronic institutions (EIs), a specific model of open, structured, regulated MAS that already has available a large collection of software tools for its specification, development, testing and execution. To address the second question we will implement a centralised service in the *Agent-based Middleware for Electronic Institutions* (AMELI) [Esteva et al., 2004] that executes a standard production system. Norms in our language will be translated into rules for that standard rule engine.

More concretely, we extend the work in [Vázquez-Salceda et al., 2004], which

proposes a theoretical deontic language, with norms that are kept active during a time interval, and conditional norms over the state of the institution, *e.g.* the observable attributes of agents. Moreover, our extension includes the possibility to sanction or reward agents by modifying their external attributes.We also provide operational semantics to the language by translating constructs written in it into Jess rules [Sandia National Labs, 2006].

As for the norm service in AMELI, governors (cf. Section 2.3.1) provide us with changes in the values of observable variables and use it to check if illocutions are finally permitted. Likewise, this service also changes the values of variables stored in governors.

3.1 A Normative Language for Electronic Institutions

As we mentioned in the beginning of the chapter, there is a need to redefine norms in electronic institutions to include time restrictions. Thus, we propose the BNF description of our normative language as follows:

$$
\begin{aligned}
NORM &:= N(\ utter(S, W, I)\ \langle TIME\rangle\ \langle \textsf{IF}\ C\rangle\ ) \\
N &:= \ \textsf{OBLIGED}\ \mid\ \textsf{PERMITTED}\ \mid \\
&\quad\ \textsf{FORBIDDEN} \\
I &:= \iota(A, R, A, R, M, T) \\
TIME &:= \textsf{BEFORE}\ D\ \mid\ \textsf{AFTER}\ D\ \mid \\
&\quad\ \textsf{BETWEEN}\ (D, D)\ \mid \\
&\quad\ \textsf{BEFORE}\ uttered(S^*, W^*, I^*)\ \mid \\
&\quad\ \textsf{AFTER}\ uttered(S^*, W^*, I^*)\ \mid \\
&\quad\ \textsf{BETWEEN}\ (\ uttered(S^*, W^*, I^*), \\
&\qquad\qquad uttered(S^*, W^*, I^*)\ ) \\
C &:= \neg\ (CONDS)\ \mid\ CONDS \\
CONDS &:= \langle\neg\,\rangle COND\ \langle, C\rangle \\
COND &:= V\ OP\ V\ \mid\ uttered(S^*, W^*, I^*)\ \mid \\
&\quad\ N(\ utter(S^*, W^*, I^*)\ )\ \mid\ predicate \\
V &:= AT\ \mid\ F\ \mid\ value \\
AT &:= identifier.attribute \mid variable \\
OP &:= \ >\mid <\mid \geq \mid \leq \mid = \\
SANCTION &:= \ \textsf{SANCTION}\ (\ (COMMS)\ \textsf{IF}\ NP\ (NORM)\ ) \\
NP &:= \ \textsf{VIOLATED}\ \mid\ \textsf{COMPLIED} \\
COMMS &:= \ COMM\langle, COMMS\rangle \\
COMM &:= \ AT\ =\ F\ \mid\ F \\
F &:= \ identifier(\ <ARGS>\ ) \\
ARGS &:= \ V\ <,\ V>
\end{aligned}
$$

where S is a scene identifier; W is a state identifier; ι is an illocutionary particle; A is an agent identifier; R is a role identifier; M is a content message in the language L_O from the dialogical framework; T is a time stamp; D is a deadline; S^*, W^*, I^*, A^*, R^*, M^*, T^* are expressions which may contain variables referring, respectively, to scenes, states, illocutions, agent identifiers, role identifiers,

messages and time stamp; and *predicate* is a first-order formula whose variables are universally quantified.

On the one hand, $utter(s^*, w^*, i^*)$ is the predicate that represents the action (not carried out yet) of submitting an illocution at the state w^* of scene s^*. This predicate is the only one that can be restricted with deontic operators. On the other hand, $uttered(s^*, w^*, i^*)$ is used to denote that the submission of an illocution has been carried out. The latter predicate can be used in the conditional construct of a normative rule.

From the BNF notation it follows that a norm (*NORM*) can be either an obligation (OBLIGED), a permission (PERMITTED) or a prohibition (FORBIDDEN) concerning the utterance of a given illocution $utter(S, W, I)$ if conditions are satisfied (IF C). The BEFORE construct is used to activate the norm before a deadline or an action. The AFTER construct is used to activate the norm after a given deadline or an action. The BETWEEN construct results from the combination of the previous two and it is used to activate the norm once the time specified by the first argument is reached and de-activate it once the time specified by the second argument is reached. The IF construct is used to introduce conditions over variables, agents' observable attributes or function results. The *AT* definition denotes how attribute values can be accessed with the language, *identifier.attribute* denotes that the value of the attribute with name *attribute* of the agent or object with name *identifier* is retrieved.

Sanctions (and, analogously, rewards) can also be expressed by defining the sequence of attribute updates or functions (*COMMS*) to be executed if a norm is violated (analogously, complied) that is, VIOLATED *NORM* or COMPLIED *NORM*.

3.1.1 Examples

In this section we show how to use our normative language through several examples. All these examples, along with the rest of examples in this chapter, are based on an electronic auction institution. The institution has four scenes or activities: *Registration*, where agents sign in along with the information about the goods they want either buy or sell; *Auction*, where the actual bidding takes place; *Payment*, where buyers pay for acquired items and sellers are paid; and *Delivery*, where the sold goods are delivered to the acquiring buyers.

```
OBLIGED   (utter(payment, W,
               inform(A, buyer, B, payee, pay(IT, P)))
BEFORE    uttered(payment, w5,
               inform(B, payee, all, buyer,
                   close()))
IF        uttered(auction, w2,
               inform(A, auctioneer, all, buyer,
                   sold(IT, P, C))),
          A.credit > P
```

Figure 3.1: Conditional obligation with a deadline

After the registration of agents and goods, agents join the *Auction* scene to

start a Dutch auction. Initially, the auctioneer agent informs all buyers about the good being auctioned along with its initial price. The auctioneer progressively decreases the call price until a bidder stops the clock. If the good has not been sold when the call price reaches the *reserve price*, the auction finishes off and the good is withdrawn. If there is a bid collision, i.e. more than one bid is submitted at the same time, the call price is increased and a new round is started. If only an agent places the bid during a round and has enough credit, it wins the auction. When an agent wins an auction it must proceed to the *Payment* scene to pay for the purchased goods. After the payment of goods, an agent taking on the *storemanager* role must deliver them to the buyer before a deadline.

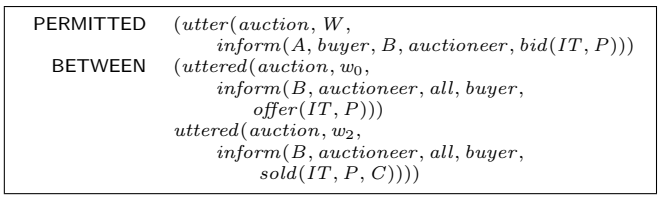

Figure 3.2: Permission in an interval of time

Figure 3.1 contains a conditional obligation a with deadline. Intuitively, it means that if a buyer submitted a winning bid for a good, he must pay for it before the payment scene is closed: If agent A, playing the *buyer* role, submits to agent C, playing the *auctioneer* role, a winning bid for a good at price P (denoted as *sold*) and agent A's credit is greater than P, then A is obliged to pay in the *payment* scene to an agent playing the *payee* role before the latter closes the scene.

Figure 3.2 shows a permission that is active during a time interval. Its intuitive meaning is that a buyer is permitted to bid after hearing an offer but before the auctioneer declares the sale: Agent A playing the *buyer* role is permitted to submit a bid for an item IT to agent B playing the *auctioneer* role in the *auction* scene after B informs buyers of an offer but before B informs buyers of the sale.

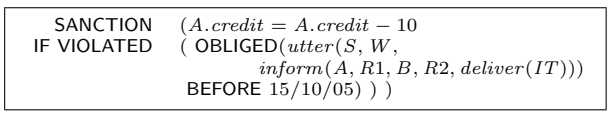

Figure 3.3: Sanction related to a deadline violation

Figure 3.3 shows a sanction on an agent's credit when a deadline cannot be met. If agent A is obliged to inform about the delivery of an item before a deadline and the agent does not meet this, his credit is reduced by ten units.

3.2 Executable Norms

Once the normative language has been defined, we need to handle the normative state of an institution. We chose a rule-based system to implement norms because the normative language is of the form *preconditions* $\rightsquigarrow$ *postconditions*, which is easily expressable with rules. In order to facilitate the integration with AMELI we decided to implement this tool with Jess since both are written in Java.

In this section we first introduce the (norm) engine used for implementing executable norms. The translation of norms expressed in the language presented in section 3.1 into executable norms written in Jess is also detailed.

3.2.1 Jess

Jess is an expert system shell and scripting language from Sandia National Laboratories [Sandia National Labs, 2006] written entirely in Java [Gossling, 1996]. Jess supports the development of rule-based systems that can be tightly coupled to code written in Java. It can manipulate Java objects and can be extended with new functions implemented in Java.

Facts

A rule-based system maintains a collection of knowledge portions called facts. This collection is known as the knowledge base. In Jess, there are three kinds of facts: ordered facts, unordered facts, and definstance facts. Ordered facts are simply Lisp-style lists where the first field, the head of the list, acts as a category for the fact. Unordered facts allow the programmer to structure the properties of a fact in slots. Before the creation of unordered facts, the slots they have must be defined using the *deftemplate* construct.

Figure 3.4 shows an example of an unordered fact template used to model the predicate *uttered*. An `uttered` fact is composed of several slots: `scene`, `state`, `agent`, `receiver`, `performative` and `content`. The scene and state where an utterance takes place is specified by the `scene` and `state` slot; while the `agent` and `receiver` slots define the sender and receiver of the message (`content`). The illocutionary particle of the illocution is stated by the `performative` slot.

```
(deftemplate uttered
    (slot scene)
    (slot state)
    (slot agent)
    (slot receiver)
    (slot performative)
    (multislot content))
```

Figure 3.4: Example of a Jess unordered fact

Rules

Rules have two parts separated by the connective "=>": a left-hand side (LHS) and a right-hand side (RHS). The LHS is employed for matching fact patterns. The RHS is a list of actions (postconditions) to perform if the patterns of the LHS (preconditions) are satisfied. These actions are typically method calls. An important feature of Jess is that the RHS can call native Jess methods, instance methods of externally referenced Java objects and static class methods. This feature adds enormous flexibility to the code.

```
(defrule cob-1-sanction
    "Reduce agent's credit on violation"
     (V  (type negative) (constraints ?c) (agent ?a)
         (scene deliver) (state w0) (receiver ?b)
         (performative inform) (content deliver ?it))
     (agent (id ?a) (attrs ?at ))
     =>
        (bind ?old (?at get "credit"))
        (bind ?new (- ?old 100))
        (?at put "credit" ?new))
```

Figure 3.5: Example of a Jess rule

Figure 3.5 shows an example of a rule. When a violation occurs, that is, when there exists a fact `V` with the specified slots and the attributes of the violator agent `?a` can be retrieved in the variable `?at` then store the value of the credit of the agent in variable `?old`, store in `?new` the value of variable `?old` decreased by a hundred and change the credit of the violator agent `?a` to the value of variable `?new`.

3.2.2 Norm implementation

In addition to the normative language we need to keep the sequence of actions done at run-time and to query what actions are permitted or forbidden and what are the pending obligations. To introduce utterances, permissions, prohibitions and obligations in the norm engine, a translation from our language to Jess rules is needed. This translation can be carried out using the criteria established in the following sections.

We define four types of Jess unordered facts: `O`, `P`, `F` and `V` that stand, respectively, for obligations, permissions, prohibitions and violations. Furthermore, we distinguish three types of norms: conditional, action-dependent and time-dependent norms.

Conditional norms.

Conditional norms are those norms that include an IF section. The translation of IF sections is directly realised by placing the conditions in the LHS of a Jess rule.

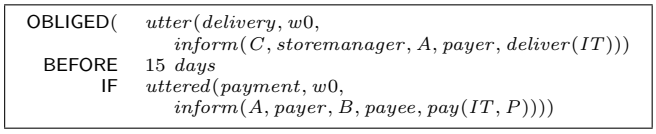

Figure 3.6: Example of a conditional obligation with a deadline

Action-dependent norms.

Action-dependent norms are those norms that include a BEFORE, AFTER or BETWEEN section followed by an action (as shown in figure 3.2). To translate an obligation to be fulfilled before the utterance of an illocution i_1, we add a rule that asserts a violation fact if illocution i_1 has been uttered but the obliged illocution has not. The assertion of facts can be achieved with the Jess function `(assert ?fact)`. The translation of permissions or prohibitions that are active before the utterance of an illocution i_1 occurs is made by asserting the given permission or prohibition and adding to the Jess engine a rule that retracts it when illocution i_1 is uttered.

In order to translate obligations, permissions and prohibitions that are active after the utterance of a given illocution i_2; we add a rule that asserts the obligation, permission or prohibition when i_2 is uttered.

The translation of permissions, prohibitions and obligations during a time interval (BETWEEN construct) is a combination of the three previous cases. We decompose the BETWEEN construct into two Jess rules as if it had an AFTER and BEFORE constructs. The translation of these constructs is carried out as stated above.

Time-dependent norms.

Time-dependent norms are those norms that include a BEFORE, AFTER or BETWEEN section followed by a date.

To translate rules with temporal restrictions (i.e. the BEFORE, AFTER and BETWEEN constructs with time objects) into Jess rules we use the user-defined function `(set-deadline ?deadline ?rule)` where ?*deadline* is an absolute date object indicating when the rule fires and ?*rule* is a string-based representation of a rule. In this way the `set-deadline` function adds the given rule to the Jess engine only when the specified absolute date arrives.

We use the `set-deadline` function to translate obligations with deadline (BEFORE construct). It adds a Jess rule that asserts a violation when the deadline has not been met. In other words, it checks, after the deadline, if the obliged illocution has not been uttered yet, in order to fire the corresponding violation.

The translation of permissions and prohibitions that are active before a deadline is done by asserting the permission or prohibition and setting a deadline rule that retracts the permission or prohibition when the deadline has passed.

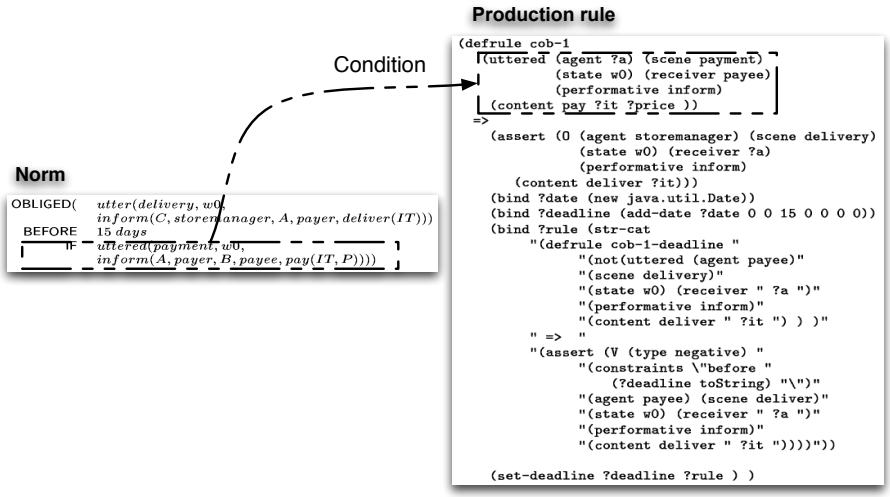

Figure 3.7: Translating the condition

Figures 3.6 to 3.9 show an example of the translation of a conditional obligation with a deadline into a Jess rule. They represent a rule establishing that paid goods must be delivered within 15 days. If agent A playing the *payer* role pays for an item to an agent B playing the *payee* role in the *payment* scene, an agent playing the *storemanager* role must deliver that item to the purchaser in the *delivery* scene within 15 days.

To translate obligations, permission or prohibitions that activate after a deadline, we add a deadline rule that asserts the obligation, permission or prohibition after the deadline.

For this purpose we use the `set-deadline` function to add a Jess rule that asserts the obligation, permission or prohibition once the deadline has passed.

Finally, obligations, permissions and prohibitions during a time interval can be translated as a combination of the previous two cases: we add a rule for the AFTER construct and another one for the BEFORE construct.

Figure 3.10 depicts the time diagram of a rule with a BETWEEN construct which is translated into two Jess rules that activate at times t_1 and t_2.

Figure 3.12 shows the translation of the normative rule 3.11 into a Jess rule. Figures 3.11 and 3.12 show a compound norm that has conditional and temporal sections. In figure 3.12 the action dependence of the norm is expressed in the conditional section. In figure 3.11 the time dependence is described by the BETWEEN construct. They oblige a store manager agent to deliver the goods between 3 to 15 days after the sale date.

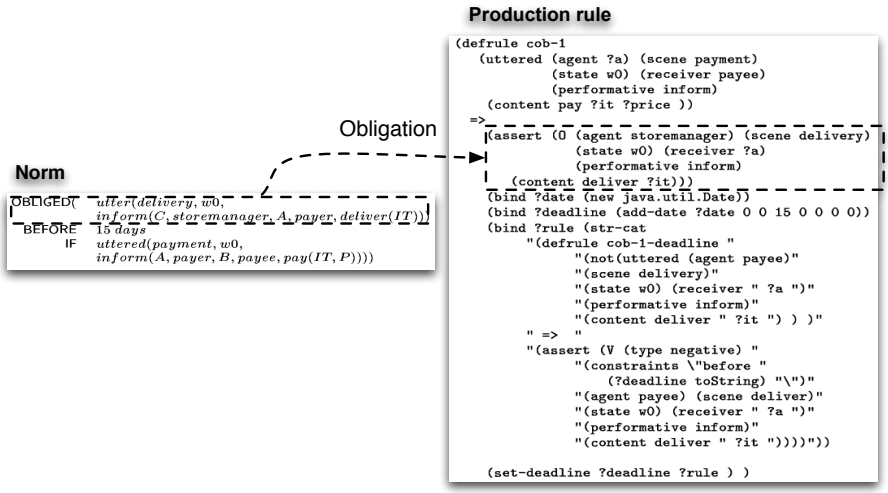

Figure 3.8: Translating the obligation

3.3 Developing and Deploying Norms

Figure 3.13 shows the process of developing norms and preparing them for execution. As any software process, after an initial analysis of the problem, we obtain a set of requirements of the problem. For instance, we can conclude that there is a certain seller that wants a given item to be auctioned. At design time, we reformulate these requirements into permissions, prohibitions and obligations over the agents. For instance, after we choose to implement the Dutch auction protocol one of the new requirements could be "The storemanager is given 15 days to deliver the paid item". This can be reformulated as "The storemanager is obliged to deliver item IT to agent A within 15 days if agent A has paid for item IT" (Fig. 3.6). Then, at development time, we translate these norms into production rules as we explained in section 3.2.2 (Cf. Figs. 3.7 - 3.9 for the translation of the norm of the example). We start the *run time* step by providing the production rules obtained at the last step into the norm engine. Finally, obligations, permissions, prohibitions, violations and illocutions are entered into the production system as facts as the MAS evolves, running the production rules and updating the facts, *i.e.* the normative state of the MAS.

3.3.1 Automatic Translation of Norms

To reduce the work in the life cycle mentioned above, we have developed a compiler that automatically translates norms written in the language presented in this chapter into Jess rules. Figure A.2 shows a UML diagram of the automatic translator. This compiler is composed of a parser-writer and a translator. The parser-writer transforms the contents of a text file, if it follows the norm lan-

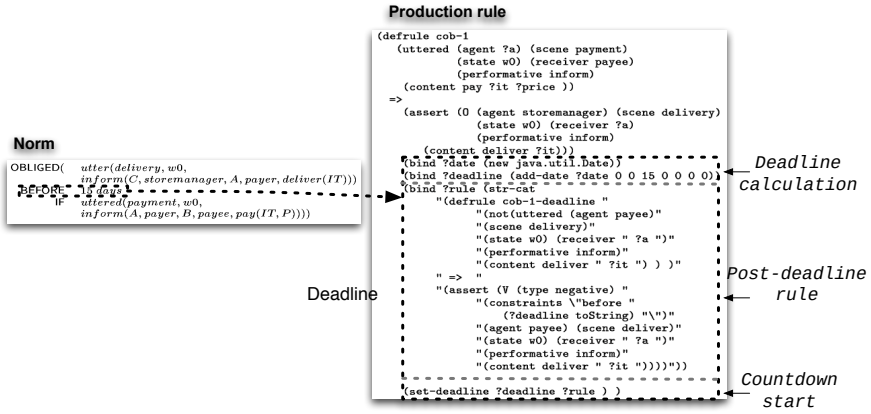

Figure 3.9: Translating the deadline

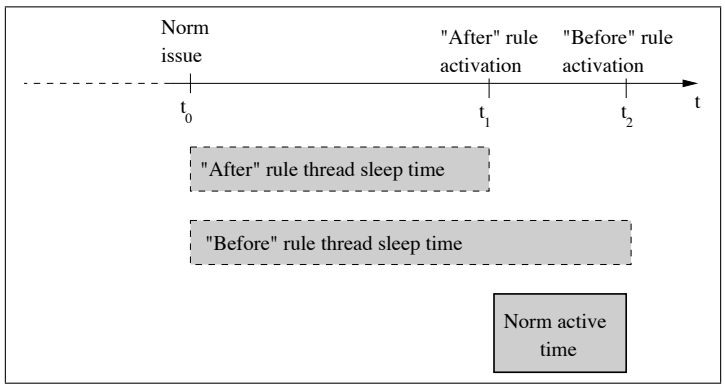

Figure 3.10: Rule activation for norm OBLIGED($utter(s, w, i)$ BETWEEN t_1, t_2)

guage syntax, into a vector of *DeonticNorm* objects and it is capable of writing *DeonticNorm* and *JessNorm* objects into strings or text files. The translator sequentially processes a norm, and then its sanctions and rewards. The processing of a *DeonticNorm* object consists in checking which constructs appear and creating a *JessNorm* object according the guidelines in Section 3.2.2.

3.3.2 Integration with Electronic Institutions

To integrate the approach presented in this chapter with electronic institutions, we need to feed the production system with facts as the MAS evolves. For that reason, AMELI, the agent middleware for electronic institutions, has been extended for this purpose. We created the *JessNormEngine* as a service in AMELI (by implementing the *Service* interface) This class also implements the interface *NormEngine* shown in Figure A.1. Whenever an illocution is uttered in

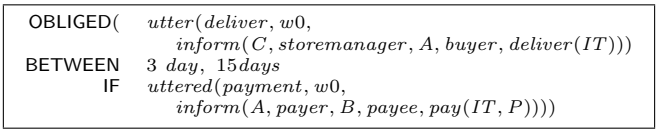

Figure 3.11: Conditional obligation along a time interval

```
(defrule obt-1
    (uttered (agent ?a) (scene payment)
             (state w0) (receiver payee)
             (performative inform)
             (content pay ?it ?price ))
    =>
      (bind ?date (new java.util.Date))
      (bind ?t1 (add-date ?date 0 0 3 0 0 0 0))
      (bind ?t2 (add-date ?date 0 0 15 0 0 0 0))

      (bind ?rule1 (str-cat
        "(defrule obt-1-after => "
        "(assert (O (agent storemanager) (scene deliver) "
        "(state w0) (receiver " ?a ")"
        "(performative inform)
        "(content deliver it))))" ))

      (bind ?rule2 (str-cat
        "(defrule obt-1-before =>"
        "(assert (V (type negative)"
        "(constraints \"before "
              (?t2 toString) "\")"
        "(agent storemanager) (scene deliver)"
        "(state w0) (receiver " ?a ")"
        "(performative inform)"
        "(content deliver it) )))" ))

      (set-deadline ?t1 ?rule1 )
      (set-deadline ?t2 ?rule2 ))
```

Figure 3.12: Implementation of a conditional obligation along a time interval

AMELI, this service queries the Jess engine if permissions and prohibitions exist for that action, if a permission exists and a prohibition does not exist then it asserts the illocution to the Jess engine. Sanction and reward rules are modified to include the Java calls necessary to modify the attributes in AMELI.

3.4 Conclusions

In this chapter we have approached how to specify norms and make them operational to regulate multi-agent activities from a centralised perspective.

As a first attempt to answer the first question, we have proposed a normative language for specifying obligations, permissions, prohibitions, violations, rewards and sanctions to restrict agents' dialogical actions. This normative language can be used in electronic institutions, a particular model of structured, open and regulated MAS, obtaining a higher degree of expressiveness and flexibility because norms can be kept active during a time interval, or may be activated by a state of the MAS. Moreover, our extension includes the possibility to sanction and reward agents by modifying their external attributes.

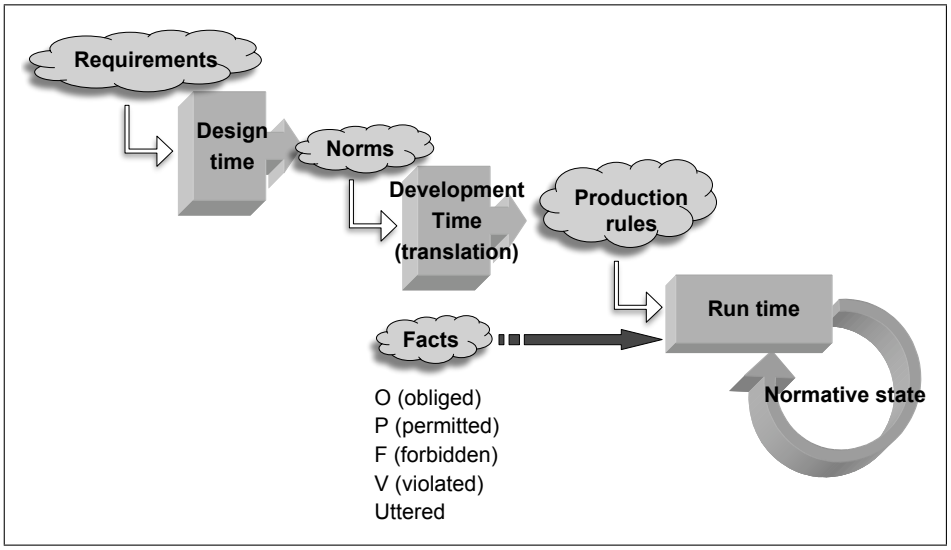

Figure 3.13: Developing and deploying norms

We provided an operational semantics to norms in electronic institutions by means of a translation of constructs of the language presented into Jess rules [Sandia National Labs, 2006] thus making them executable by means of the Jess rule engine.

As a first attempt to answer the second question, we have implemented a norm service in AMELI that maintains the normative state of an institution, *i.e* the permissions, prohibitions and pending obligations that hold in the current state of execution.

Although Jess rules were proposed in [Esteva, 2003] to be dealt with by each governor (G_i in Fig. 3.14) and therefore distributable, the norm engine presented in this chapter is centralised as it is implemented as a service running for the whole institution. We have noticed that some norms cannot be dealt completely isolated from other governors. For instance, a prohibition on all the buyers to make an unsupported bid can be enforced by each governor of buyer agents. However, a prohibition on two agents to perform certain simultaneous actions cannot be enforced by governors as they do not have access to actions of other agents. Our goal is to implement distributed enforcement of these latter kind of norms. To achieve this, in the next two chapters we propose regulation at activity level, where actions of all participant agents are known.

Although the proposals of this chapter answer the first two research questions posed in Chapter 1, we noticed that we can provide more expressiveness to candidate languages by specifying norms and making them operational but including further constraints than just temporal ones. For instance, in our wire factory example, we may specify that copper sellers are expected to deliver between 100 and 200 kilograms of copper by the end of the day. If we assume now that *WireMaking Ltd.* has minimum requirements of prime materials that

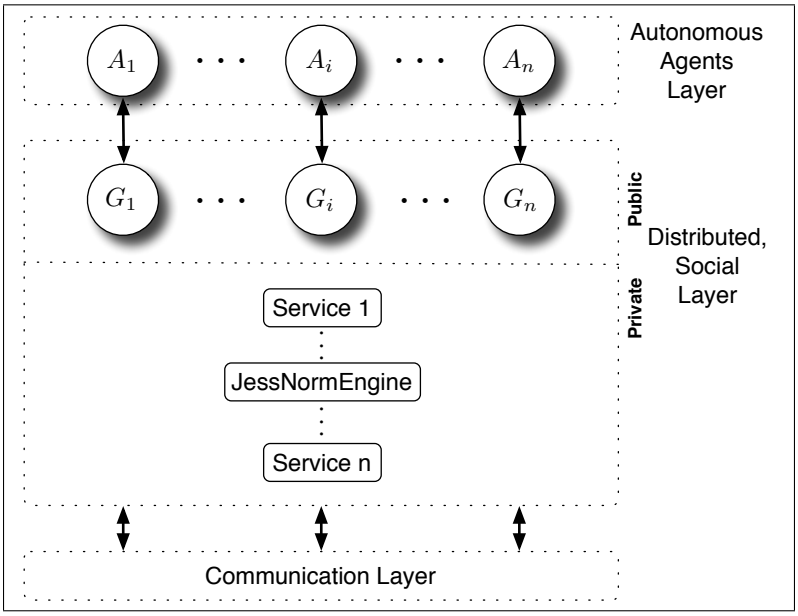

Figure 3.14: AMELI and the JessNormEngine Service

it consumes each day and a maximum quantity of copper that it can store, then we may also want to specify that copper sellers are prohibited to deliver less than 0.75 and more than 1.25 times the stipulated amount of copper for today. Thus, in the following chapter we include constraint management in our proposal.

Chapter 4

Constraint-based Regulation

In this chapter we propose a rule language to specify norms and arithmetical constraints, we provide a computational model that allows the processing of simultaneous speech acts and we present an implementation of the interpreter for the language we propose.

In the previous chapter we started answering our research questions by extending electronic institutions with a richer notion of norm that includes obligations, permissions and prohibitions with temporal constraints, sanctions when a violation occurs and rewards when norms are complied. However, norms may have further constraints associated besides temporal ones. For instance, in a scenario wherein a selling agent is obliged to deliver a product satisfying some quality requirements before a deadline, both the quality requirements and the delivery deadline can be regarded as constraints that must be considered as part of the agent's obligation. Thus, only when the agent delivers the good satisfying all the constraints, is that we should regard the obligation as fulfilled. We also notice that since the deadline might eventually be changed, we also require the capability of modifying constraints at run-time. For this reason and to reduce the complexity of the rules, we decided to create a rule language without forward chaining for the management of predicates with constraints.

Despite the fact that Jess rules are proposed in [Esteva, 2003] to be dealt with by each governor (G_i in Fig. 3.14) we have noticed that some norms cannot be dealt completely isolated from other governors. For instance, a prohibition on all the buyers to make an unsupported bid can be enforced by each governor representing an agent who is a buyer. However, a prohibition on two agents to perform certain simultaneous actions cannot be enforced by governors as they do not have access to actions of other agents. Our goal is to implement the distributed enforcement of this latter kind of norms. To achieve this, in this chapter and in the following ones we propose regulation at activity level, where the actions of all participant agents are known.

In this chapter we present *IRL*, the Institutional Rule Language. Constraints are entities managed explicitly in conjunction with predicates – we accommodate, as we show, constraints in our semantics using standard constraint solving techniques. Constraints allow for more sophisticated notions of norms and normative positions to be expressed. For instance, in a scenario in which a selling agent is obliged to deliver a product satisfying some quality requirements before a deadline, both the quality requirements and the delivery deadline can be regarded as constraints that must be considered as part of the agent's obligation. Thus, when the agent delivers the good satisfying all the constraints, we should regard the obligation as fulfilled. Notice too that since the deadline might eventually be changed, we also require the capability of modifying constraints at run-time.

We also show how agent's behaviour are regulated by means of protocols and norms that can be represented and be given a computational realisation. By regulating interactions with rules, we provide a light-weight engine in order to regulate open MAS.

This language has been conceived to represent distinct flavours of deontic notions and relationships: we can define different situations in which different deontic notions hold, that is, we can define the behaviour of the institution in certain conditions. In our language, we can specify situations in which, for instance, prohibitions cannot be violated and situations in which, *e.g.*, prohibitions of certain actions can be violated under penalties.

Our normative approach gives more flexibility to EIs [Esteva, 2003] in that we can also capture deviant behaviour. Our work sets the foundations to specify and implement light-weight institutions via rules.

The main contributions of this chapter are:

1. a means to specify what an agent can, may, may not and ought to utter using normative positions and constraints;

2. an operational semantics to the above mentioned specification by means of rules and constraint solving techniques;

3. a computational model that establishes how agents' speech acts are processed; and

4. the application of this computational notion of norm to implement and enrich a model of regulated MAS like Electronic Institutions and, as an illustrative example, its application to regulate the Dutch Auction.

4.1 A Rule Language for Managing Normative Positions

In this section we present the work introduced in [García-Camino et al., 2006a], a rule-based language for the specification of the translation of agents' illocutions into actions in the normative environment. We consider that agents can (directly

or indirectly) cause changes in their own normative positions (*e.g.*, by bidding in an auction), in the normative positions of other agents (*e.g.*, by delegating or commanding), in the observable attributes of agents (*e.g.*, "badmouthing" an agent can decrease its reputation), or in the state of resources of the environment (*e.g.*, moving a box changes its location). By *environment* we mean the shared resources which are not part of the agents and, therefore, cannot be freely accessed and modified. By *state of affairs* we mean the normative environment, *i.e.* the representation of aspects of the MASs enactment including the set of attributes that a community of agents can access or modify in an unregulated setting.

In regulated MASs these attributes can only be accessed and modified under certain conditions. Our rule-based language allows us to represent regulated changes fulfilling the requirement that a normative language should be declarative. The rules depict how the state of affairs changes as agents interact with each other or the environment.

We make use of the closed world assumption (CWA) [Apt, 1997] since we assume a MAS as a communication middleware that manages (and has access to) all protocols it may regulate. Therefore, we consider as false all formulae not included in the state of affairs of a MAS since we cannot regulate them.

We now introduce an example of enactment of our computational model using a Dutch Auction scenario. There are some goods that are expected to be sold to one of the agents participating in the auction. We consider these goods part of a real environment but represented in the normative environment. However, they are owned by one agent, enacting the role of seller, until the auctioneer, a special kind of agent that regulates the auction, finishes the process determining a winner and the latter pays for the auctioned goods. As the state of affairs, *i.e.* as the normative environment of the auction, we consider the current credit of the participants, the ownership of the goods that are part of the environment and the history of speech acts that have been considered valid at some point of the enactment of the auction. For instance, whenever the auctioneer offers a good for a given price, it has to be checked that the illocution was uttered in the correct point in the protocol by applying the rules. If this is the case, this illocution is added to the state of affairs for later checking. Continuing with the example, the participant agents may now bid for the item at the offered price. These attempts are added to the previous state of affairs giving rise to the current state of affairs before applying the rules to check which attempts are valid. After the application of the rules, only the valid bids remain in the state of affairs, e.g. those that the agents may afford. If it is the case that there is only one bid, a winner may be determined and the obligation of the winner to pay for the goods arises.

Figure 4.1 depicts the computational model we propose. An initial state of affairs Δ_0 (possibly empty) is offered (represented by "$\Rrightarrow$") to a set of agents $(ag_1, \cdots, ag_n)$. These agents can add their illocutions $(\Xi_1^0, \cdots, \Xi_n^0)$ to the state of affairs (via "$\updownarrow$"). Ξ_i^t is the (possibly empty) set of illocutions added by agent i at state of affairs Δ_t. After an established amount of time, we perform an

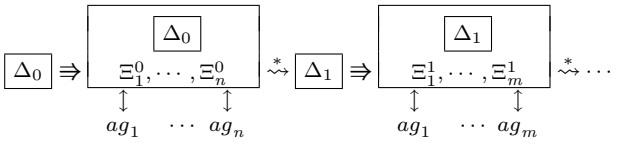

Figure 4.1: Semantics as a Sequence of Δ's

exhaustive application of rules (denoted by "$\overset{*}{\rightsquigarrow}$") to the modified state, yielding a new state of affairs Δ_1. This new state will, on its turn, be offered to the agents for them to add their events, and the same process will go on.

4.1.1 Preliminary Definitions

We initially define some basic concepts. The building blocks of our language are *terms*:

Definition 4.1.1. A term, denoted as τ, is

- Any variable x, y, z (with or without subscripts) or
- Any construct $f^n(\tau_1, \ldots, \tau_n)$, where f^n is an n-ary function symbol and $\tau_1, \ldots, \tau_n$ are terms.

Terms f^0 stand for *constants* and will be denoted as a, b, c (with or without subscripts). We shall also make use of numbers and arithmetic functions to build our terms; arithmetic functions may appear infix, following their usual conventions. We adopt Prolog's convention [Apt, 1997] using strings starting with a capital letter to represent variables and strings starting with a small letter to represent constants. Some examples of terms are *Price* (a variable) and $send(a, B, Price \times 1.2)$ (a function). We also need to define *atomic formulae*:

Definition 4.1.2. An atomic formula, denoted as α, is any construct $p^n(\tau_1, \ldots, \tau_n)$, where p^n is an n-ary predicate symbol and $\tau_1, \ldots, \tau_n$ are terms.

When the context makes it clear what n is we can drop it. p^0 stands for propositions. We shall employ arithmetic relations (*e.g.*, $=$, $\neq$, and so on) as predicate symbols, and these will appear in their usual infix notation. We also make use of atomic formulae built with arithmetic relations to represent *constraints* on variables – these atomic formulae have a special status, as we explain below. We give a definition of our constraints, a subset of atomic formulae:

Definition 4.1.3. A constraint γ is a binary atomic formula $\tau \lhd \tau'$, where $\lhd \in \{=, \neq, >, \geq, <, \leq\}$.

We shall use $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ as a set of constraints. A state of affairs is a set of atomic formulae, representing (as shown below) the normative positions of agents, observable agent attributes and the state of the environment[1].

Definition 4.1.4. A state of affairs $\Delta = \{\alpha_0 : \Gamma_0, \ldots, \alpha_n : \Gamma_n\}$ is a a finite and possibly empty set of implicitly, universally quantified atomic formulae α_i restricted by a possibly empty set of constraints Γ_i, $0 \leq i \leq n$. When the set of constraints is empty, we write just α_i.

4.1.2 A Language for Rules with Constraints

Our rules are constructs of the form $LHS \rightsquigarrow RHS$, where LHS contains a representation of parts of the current state of affairs which, if they hold, will cause the rule to be triggered. RHS describes the updates to the current state of affairs, yielding the next state of affairs:

Definition 4.1.5. A rule, denoted as R, is defined as:

$$
\begin{array}{rcl}
\mathsf{R} & ::= & LHS \rightsquigarrow RHS \\
LHS & ::= & LHS \ \texttt{\&} \ LHS \mid LHS \ \texttt{||} \ LHS \mid \texttt{not}(LHS) \mid \mathsf{Lit} \\
RHS & ::= & \mathsf{U}, RHS \mid \mathsf{U} \\
\mathsf{Lit} & ::= & \mathsf{Atf} \mid \texttt{sat}(\Gamma) \mid x = \{\mathsf{Atf} \mid LHS\} \\
\mathsf{Atf} & ::= & \alpha \mid \alpha : \Gamma \\
\mathsf{U} & ::= & \texttt{add}(\mathsf{Atf}) \mid \texttt{del}(\mathsf{Atf})
\end{array}
$$

where x is a variable name and Γ is a list of constraints or a variable name.

Intuitively, the left-hand side LHS describes the conditions the current state of affairs ought to have for the rule to apply. The right-hand side RHS describes the updates to the current state of affairs, yielding the next state of affairs.

In the next section we define the semantics of each construct above, but intuitively, the constructs α and $\alpha : \Gamma$ check whether there exists an atomic formula in the state of affairs matching the atomic formulae α or an atomic formula with constraints, $\texttt{sat}(\Gamma)$ checks whether Γ (a set of constraints) is satisfiable. We also make use of a special kind of term, called a *set constructor*, represented as $\{\mathsf{Atf} \mid LHS\}$. This construct is useful when we need to refer to all atomic formulae in the state of affairs (Atfs) for which LHS holds. For instance, $\{(p(A, B, C) \mid B > 20 \ \texttt{\&} \ C < 100\}$ stands for the set of atomic formulae $p(A, B, C)$ such that B is greater than 20 and C is less than 100. Notice that if we want to check whether variable B in $p(A, B, C)$ has a constraint representing a lower bound in value V, we may use the expression $\{p(A, B, C) : Cs \mid (B > V) \in Cs\}$. The Us represent updates: they add to the state of affairs (via operator `add`) or remove from the state of affairs (via operator `del`) atomic formulae.

[1] We refer to the *state of the environment* as the subset of atomic formulae representing observable aspects of the environment in a given point in time.

4.1.3 Semantics of Rules

As shown in figure 4.1, we define the semantics of our rules as a relationship between states of affairs: rules map an existing state of affairs to a new state of affairs. In this section we define this relationship. In the definitions below we rely on the concept of *substitution*, that is, the set of values for variables in a computation [Apt, 1997, Fitting, 1990]:

Definition 4.1.6. A substitution $\sigma = \{x_0/\tau_0, \ldots, x_n/\tau_n\}$ is a finite and possibly empty set of pairs x_i/τ_i, $0 \leq i \leq n$.

Definition 4.1.7. The application of a substitution to an atomic formulae α possibly restricted by a set of constraints $\{\gamma_0, \ldots, \gamma_m\}$ is as follows:

1. $c \cdot \sigma = c$ for a constant c;
2. $x \cdot \sigma = \tau \cdot \sigma$ if $x/\tau \in \sigma$; otherwise $x \cdot \sigma = x$;
3. $p^n(\tau_0, \ldots, \tau_n) \cdot \sigma = p^n(\tau_0 \cdot \sigma, \ldots, \tau_n \cdot \sigma)$;
4. $p^n(\tau_0, \ldots, \tau_n) : \{\gamma_0, \ldots, \gamma_m\} \cdot \sigma = p^n(\tau_0 \cdot \sigma, \ldots, \tau_n \cdot \sigma) : \{\gamma_0 \cdot \sigma, \ldots, \gamma_m \cdot \sigma\}$.

Definition 4.1.8. The application of a substitution to a sequence is the sequence of the application of the substitution to each element: $\langle \alpha_1, \ldots, \alpha_n \rangle \cdot \sigma = \langle \alpha_1 \cdot \sigma, \ldots, \alpha_n \cdot \sigma \rangle$

We now define the semantics of the LHS of a rule, that is, how a rule is triggered:

Definition 4.1.9. $\mathbf{s}_l(\Delta, LHS, \sigma)$ holds between state Δ, the left-hand side of a rule LHS and a substitution σ depending on the format of LHS:

1. $\mathbf{s}_l(\Delta, LHS \ \texttt{\&} \ LHS', \sigma)$ holds iff $\mathbf{s}_l(\Delta, LHS, \sigma')$ and $\mathbf{s}_l(\Delta, LHS' \cdot \sigma', \sigma'')$ hold (in this order) and $\sigma = \sigma' \cup \sigma''$.
2. $\mathbf{s}_l(\Delta, LHS \ \texttt{||} \ LHS', \sigma)$ holds iff $\mathbf{s}_l(\Delta, LHS, \sigma)$ or $\mathbf{s}_l(\Delta, LHS, \sigma)$ hold.
3. $\mathbf{s}_l(\Delta, \texttt{not}(LHS), \sigma)$ holds iff $\mathbf{s}_l(\Delta, LHS, \sigma)$ does not hold.
4. $\mathbf{s}_l(\Delta, \texttt{sat}(\Gamma), \sigma)$ holds iff $satisfiable(\Gamma \cdot \sigma)$ hold.
5. $\mathbf{s}_l(\Delta, x = \{\mathsf{Atf} \mid LHS\}, \sigma)$ holds iff $\sigma = \{x/\{\mathsf{Atf} \cdot \sigma_1, \ldots, \mathsf{Atf} \cdot \sigma_n\}\}$ for the largest $n \in \mathbb{N}$ such that $\mathbf{s}_l(\Delta, \mathsf{Atf} \ \texttt{\&} \ LHS, \sigma_i), 1 \leq i \leq n$
6. $\mathbf{s}_l(\Delta, \mathsf{Atf}, \sigma)$ holds iff $\mathsf{Atf} \cdot \sigma \in \Delta$ or $\mathsf{Atf} \cdot \sigma$ holds.

Cases 1 and 2 depict the semantics of atomic formulae and how their individual substitutions are combined to provide the semantics for a conjunction and a disjunction respectively. Case 3 introduces the negation by failure. Recall that we make use of the closed world assumption. Case 4 holds if the set of constraints Γ is satisfiable; the substitution σ obtained so far, that is applied to Γ will hold an assignment of variables in a Constraint Satisfaction Problem

[Tsang, 1993]. Case 5 specifies the semantics for *set constructors*: x is the set of atomic formulae that satisfy the conditions of the set constructor. Case 6 holds when an atomic formulae (a predicate or constraint) is part of the state of affairs or it holds in the underlying programming language. It is worth noticing that, from case 1 above, the order in which conjuncts appear on the left-hand side is relevant. Our rules are means to define a deterministic program, hence the order of commands is essential.

We now define the semantics of the RHS of a rule:

Definition 4.1.10. Relation $\mathbf{s}_r(\Delta, RHS, \Delta')$ mapping a state Δ, the right-hand side of a rule RHS and a new state Δ' is defined as:

1. $\mathbf{s}_r(\Delta, (\mathsf{U}\,\mathbf{,}\,RHS), \Delta')$ holds iff both $\mathbf{s}_r(\Delta, \mathsf{U}, \Delta_1)$ and $\mathbf{s}_r(\Delta_1, RHS, \Delta')$ hold.
2. $\mathbf{s}_r(\Delta, \mathtt{add}(\alpha), \Delta')$ holds iff $\Delta' = \Delta \cup \{\alpha\}$.
3. $\mathbf{s}_r(\Delta, \mathtt{add}(\alpha : \Gamma), \Delta')$ holds iff $satisfiable(\Gamma)$ hold and $\Delta' = \Delta \cup \{\alpha : \Gamma\}$.
4. $\mathbf{s}_r(\Delta, \mathtt{add}(\alpha : \Gamma), \Delta)$ holds iff $satisfiable(\Gamma)$ does not hold.
5. $\mathbf{s}_r(\Delta, \mathtt{del}(\mathsf{Atf}), \Delta')$ holds iff $\Delta' = \Delta \setminus \{\mathsf{Atf}\}$.

Case 1 decomposes a conjunction and builds the new state by merging the partial states of each update. Case 2 caters for the insertion of atomic formulae which are not restricted with constraints. Case 3 defines how a constrained predicate is added to a state Δ: the new constraints are checked whether it can be satisfied and then it is added to Δ'. Otherwise, case 4 maintains the initial state Δ unaltered. Case 5 caters for the removal of atomic formulae (both constrained and non-constrained). We note that, from case 1 above, the order in which conjuncts appear on the right-hand side is also relevant.

Relation $\mathbf{s}_r$ applies all the updates in the RHS of one firing rule. However, we also need to apply all the updates in the $RHSs$ of all the firing rules. Relation $\mathbf{s}'_r$ applies all the updates in *UpdateList* (that proceed from the $RHSs$ of the firing rules) using $\mathbf{s}_r$.

Definition 4.1.11. Relation $\mathbf{s}'_r(\Delta, \mathit{UpdateList}, \Delta')$ mapping a state of affairs Δ, a list of updates and a new state of affairs holds iff

1. $\mathit{UpdateList} = \langle\rangle$ and $\Delta' = \Delta$; or
2. $\mathit{UpdateList} = \langle u_1, \cdots, u_n\rangle$, $\mathbf{s}_r(\Delta, u_1, \Delta'')$ and $\mathbf{s}'_r(\Delta'', \langle u_2, \cdots, u_n\rangle, \Delta')$ hold.

To complete the definition of our system, we define the semantics of our rules as relationships between states of affairs: rules map an existing state of affairs to a new state of affairs, thus modelling transitions between states of affairs. Relation $\mathbf{s}^*$ calculates the new state of affairs Δ' from an initial state Δ and a sequence of rules.

Definition 4.1.12. Relation $\mathbf{s}^*(\Delta, \mathit{Rules}, \Delta')$ mapping a state of affairs Δ, a list of rules, and a new state of affairs holds iff Us is the largest set of updates $U = RHS \cdot \sigma$ of all rules $r \in \mathit{Rules}$ of the form $r = LHS \rightsquigarrow RHS$ such that $\mathbf{s}_l(\Delta, LHS, \sigma)$, and $\mathbf{s}'_r(\Delta, Us, \Delta')$ hold.

Notice that we collect a list of the updates of the firing rules (with all the different values of σ that make $\mathbf{s}_l(\Delta, LHS, \sigma)$ hold) and then we apply the collected updates sequentially. Additionally, we do not apply forward chaining of rules, making our approach different from standard production systems as Jess, the one used in chapter 3.

4.1.4 An Interpreter for Rules with Constraints

The semantics above provide a basis for the implementation of a rule interpreter. Although we have implemented it with SICStus Prolog [SICS, 2006] we show how a rule is interpreted in figure 4.2 as a logic program, interspersed with built-in Prolog predicates; for easy referencing, we show each clause with a number on its left.

1. $\mathbf{s}_l(\Delta, (LHS \ \texttt{\&} \ LHS'), \sigma) \leftarrow \mathbf{s}_l(\Delta, LHS, \sigma'), \mathbf{s}_l(\Delta, LHS' \cdot \sigma', \sigma''), \sigma = \sigma' \cup \sigma''$
2. $\mathbf{s}_l(\Delta, (LHS \ \texttt{||} \ LHS'), \sigma) \leftarrow \mathbf{s}_l(\Delta, LHS, \sigma)$
3. $\mathbf{s}_l(\Delta, (LHS \ \texttt{||} \ LHS'), \sigma) \leftarrow \mathbf{s}_l(\Delta, LHS', \sigma)$
4. $\mathbf{s}_l(\Delta, \texttt{not}(LHS), \sigma) \leftarrow \neg \mathbf{s}_l(\Delta, LHS, \sigma)$
5. $\mathbf{s}_l(\Delta, \texttt{sat}(\Gamma), \sigma) \leftarrow satisfiable(\Gamma \cdot \sigma)$
6. $\mathbf{s}_l(\Delta, x = \{\mathsf{Atf} \mid LHS\}, \{x/AllAtfs\}) \leftarrow$
   $\texttt{findall}(\mathsf{Atf} \cdot \sigma, \mathbf{s}_l(\Delta, \mathsf{Atf} \ \texttt{\&} \ LHS, \sigma), AllAtfs)$
7. $\mathbf{s}_l(\Delta, \mathsf{Atf}, \sigma) \leftarrow \texttt{member}(\mathsf{Atf} \cdot \sigma, \Delta)$
8. $\mathbf{s}_l(\Delta, \alpha, \sigma) \leftarrow \texttt{call}(\alpha \cdot \sigma)$
9. $\mathbf{s}_r(\Delta, (\mathsf{U}\texttt{,}RHS), \Delta'') \leftarrow \mathbf{s}_r(\Delta, \mathsf{U}, \Delta'), \mathbf{s}_r(\Delta', RHS, \Delta'')$
10. $\mathbf{s}_r(\Delta, \texttt{add}(\alpha), \Delta') \leftarrow \Delta' = \Delta \cup \{\alpha\}$
11. $\mathbf{s}_r(\Delta, \texttt{add}(\alpha : \Gamma), \Delta') \leftarrow satisfiable(\Gamma), \ \Delta' = \Delta \cup \{\alpha : \Gamma\}$
12. $\mathbf{s}_r(\Delta, \texttt{add}(\alpha : \Gamma), \Delta) \leftarrow$
13. $\mathbf{s}_r(\Delta, \texttt{del}(\mathsf{Atf}), \Delta') \leftarrow \texttt{delete}(\Delta, \mathsf{Atf}, \Delta')$
14. $\mathbf{s}'_r(\Delta, [\,], \Delta') \leftarrow \Delta = \Delta'$
15. $\mathbf{s}'_r(\Delta, [U \mid Us], \Delta') \leftarrow \mathbf{s}_r(\Delta, U, \Delta''), \mathbf{s}'_r(\Delta''', Us, \Delta')$
16. $\mathbf{s}^*(\Delta, Rules, \Delta') \leftarrow$
    $\texttt{findall}(RHS, (\texttt{member}((LHS \rightsquigarrow RHS), Rules), \mathbf{s}_l(\Delta, LHS), Us),$
    $\mathbf{s}'_r(\Delta, Us, \Delta')$

Figure 4.2: Interpreter for Rules with Constraints

For each rule, we apply $\mathbf{s}_l(\Delta, LHS, \sigma)$ sequentially for all the different substitutions σ in the state of affairs and we apply and $\mathbf{s}_r(\Delta, RHS \cdot \sigma, \Delta')$ to all the RHS such that its LHS holds, *i.e.* $\mathbf{s}_l(\Delta, LHS, \sigma)$ holds. Clauses 1-8, 9-13, 14-15 and 16 are, respectively, adaptations of the cases depicted in Defs. 4.1.9-4.1.12.

We can define $satisfiable/2$ via the built-in `call_residue`$/2$ predicate, available in SICStus Prolog:

$$satisfiable(\{\gamma_1, \ldots, \gamma_n\}) \leftarrow \texttt{call_residue}((\gamma_1, \ldots, \gamma_n), _)$$

It is worth mentioning that in the actual Prolog implementation, substitutions σ appear implicitly as values of variables in terms – the logic program above looks neater when we incorporate this.

4.1.5 Pragmatics of Rules with Constraints

In this section we illustrate the pragmatics of our rules with some examples:

$$\left(\begin{array}{l} do(A, pay(P, B), T)\ \texttt{\&} \\ credit(B, X)\ \texttt{\&}\ X_2 = X + P \end{array}\right) \rightsquigarrow \left(\begin{array}{l} \texttt{del}(do(A, pay(P, B), T)), \\ \texttt{del}(credit(B, X)), \\ \texttt{add}(credit(B, X_2)) \end{array}\right) \tag{4.1}$$

$$\left(\begin{array}{l} do(A, ext(obl(X, T_2) : C, D), T)\ \texttt{\&} \\ (T < D)\ \texttt{\&}\ (T_2 < D_2) \in C\ \texttt{\&} \\ delete((T_2 > D_2), C, C')\ \texttt{\&} \\ append(C', (T_2 > D), C'') \end{array}\right) \rightsquigarrow \left(\begin{array}{l} \texttt{del}(do(A, ext(obl(X, T_2) : C, D), T)), \\ \texttt{del}(obl(X, T_2) : C), \texttt{add}(obl(X, T_2) : C'') \end{array}\right) \tag{4.2}$$

$$\left(\begin{array}{l} do(C, pay(P, B), T)\ \texttt{\&}\ min(D)\ \texttt{\&} \\ time(T)\ \texttt{\&}\ (P > D) \end{array}\right) \rightsquigarrow \left(\begin{array}{l} \texttt{del}(do(C, pay(P, B))), \\ \texttt{add}(done(C, pay(P, B), T)) \end{array}\right) \tag{4.3}$$

The first example shows a rule depicting the circumstances under which it should be applied: if agent A generates the event of paying price P to agent B and the credit of the latter is X. It also shows on the RHS the updates to perform: we ensure the event is "consumed" (thus not triggering off the rule indefinitely) and the credit of agent B is updated to $X + P$.

The second example illustrates the management of constraints: these can be manipulated like ordinary elements of the state of affairs. In that example, we show that events of type *obl* (i.e. an obligation) may have associated constraints. Particularly, this rule states that if an event of extending (*ext*) the deadlines of all the obligations to time D occurs before the deadline D and there exists a constraint restricting the time of fulfillment of the obligations to be less than a deadline $D2$, then the event is consumed, the old obligation is removed and an obligation with the new deadline is added.

The third example illustrates how constraints can additionally be *checked* for their satisfaction: when an event of paying price P is performed by agent C and there is a formula $min(D)$ (storing a minimum price), we check that all the constraints in the state of affairs, including the one establishing that the amount paid is greater than the minimum ($P > D$), are satisfied. If this is so, then we remove the event and add a record of this case to the state of affairs. Notice that we make use of a built-in predicate $time/1$ to check the current time of the system.

Our rules manage states of affairs, adding or removing formulae (expressed on the RHS) when certain conditions (expressed on the LHS) hold. As illustrated in figure 4.1, our approach accommodates the participation of agents: they add atomic formulae onto the current state of affairs – these formulae represent agent-related events, represented above as $do(Ag, Ev, T)$ that, together with further elaboration on the circumstances, will trigger off rules to update the state of affairs.

The language that we propose defines a rule-based system enhanced with constraint satisfaction techniques in order to check how specified constraints affect the facts they constrain. We have obtained a language to express, manage, check fulfilment and/or sanction unfulfilled normative positions, i.e. obligations,

permissions and prohibitions, that are bounded with constraints. Thus, the language is useful to predict a future state of affairs given an initial state and a sequence of sets of events that occur and modify the intermediate states of affairs until we reach the final one. The limitations of the language are determined by the rule engine. These limitations include the inability to plan, i.e. determine the sequence of sets of events that must occur in order to reach a given state of affairs from a given initial state, or post-dicting, i.e. determine the previously unknown facts in a partial initial state given a final state and the sequence of sets of events that have occurred. However, the goal of the language is to regulate a MAS and keep track of its evolution by prediction. Post-diction and planning would be interesting for a language that an agent could use for deciding which action to perform but this is not the aim of this thesis.

There are further concerns to be taken into account when designing rules. Clearly, what we choose to go in the state of affairs has an immediate influence as to what should appear in rules. Another concern is how we choose to represent events generated by agents. We show in the next section a representation proposal that includes information on who caused the event, the time, and a suitable description of the event.

4.2 Programming Institutional Rules

In this section we use the language presented in the previous section to enact and regulate agent behaviour in an open MAS. We assume that agent behaviour is formed by events generated by agents that the system may consider valid. We say the system *institutionalise* these events.

For illustrative purposes, we define a particular kind of event, the illocutions, used in the communication of agents in the forthcoming examples.

Definition 4.2.1. Illocutions I are ground atomic formulae $ill(p, ag, r, ag', r', \tau)$ where

- p is an element of a set of illocutionary particles (*e.g.*, *inform*, *request*, etc.).
- ag, ag' are agent identifiers.
- r, r' are role labels.
- τ, an arbitrary ground term, is the actual content of the message, built from a shared content language.

Sometimes it is useful to refer to illocutions that are not fully ground, that is, they may have uninstantiated (free) *variables* within themselves – in the description of a protocol, for instance, the precise values of the message exchanged can be left unspecified. During the enactment of the protocol agents will produce the actual values which will give rise to a (ground) illocution. We can thus define *illocution schemes*:

Definition 4.2.2. An illocution scheme $\bar{\mathsf{I}}$ is any atomic formula $ill(p, ag, r, ag', r', \tau)$ whose terms are either variables or may contain variables.

4.2.1 Institutional States

An institutional state is a state of affairs that stores all the events institutionalised during the execution of a MAS, also keeping a record of the state of the environment, all observable attributes of agents and all obligations, permissions and prohibitions associated with the agents, *i.e.* their normative positions.

We differentiate eight kinds of atomic formulae in our institutional states Δ, with the following intuitive meanings:

1. $oav(o, a, v)$ – object (or agent) o has an attribute a with value v.
2. $att(e)$ – an agent performed event e attempting to get it institutionally accepted.
3. $inst(e, t)$ – e was accepted as an institutional event at time t.
4. $old_state(w, t)$ – the execution of the scene reached state w at time t.
5. $state(w, t)$ – the execution of the scene is in state w since time t.
6. $obl(e, t_{inst}, t)$ – e *ought* to be institutionalised at time t_{inst} since t.
7. $per(e, t_{inst}, t)$ – e is *permitted* to be institutionalised at time t_{inst} since t.
8. $prh(e, t_{inst}, t)$ – e is *prohibited* to be institutionalised at time t_{inst} since t.

States of affairs, such as institutional states, and states of a protocol are related concepts but should not be confused. In a MAS, it is possible to have many instances of protocols (or various instances of the same protocol) simultaneously enacted by agents. This means that at any one time, we could have many states of protocols represented by atomic formulae in a state of affairs.

Remember that, for illustrative purposes, we focus on a particular kind of event, that is, the illocutions. Notice that, since illocutions are uttered towards another specific agent, normative positions over illocutions also are with respect to another agent, i.e. an agent may be obliged to say something to another agent.

We differentiate between brute acts, that is, events that are attempted to be accepted (*att*) and accepted events or institutional acts (*inst*). Since we aim at heterogeneous agents whose behaviour we cannot guarantee, we create a "sandbox" where agents can utter whatever they want (via *att* formulae). However, not every events that agents generate may be in accordance with the rules – the illegal events may be discarded and/or may cause sanctions, depending on the deontic notions we want or need to implement. The *inst* formulae are thus *confirmations* of the *att* formulae.

We only allow fully ground attributes, events and state control formulae (cases 1-5 above) to be present, however, in formulae 6–8 t_{inst} may be a variable

and e may contain variables. We shall use formula 4 to represent state change in a protocol in relation to global time passing. We shall use formulae 6–8 above to represent the normative positions of agents within MASs.

We do not "hardwire"deontic notions in our semantics: formulae 6-8 above represent deontic operators but not their relationships. These are captured with rules as we show in the next section. We show in figure 4.3 a sample institutional state. The utterances show a portion of the dialogue between a buyer agent and

$$\Delta = \left\{ \begin{array}{l} inst(ill(inform, ag_4, seller, ag_3, buyer, offer(car, 1200)), 10), \\ inst(ill(inform, ag_3, buyer, ag_4, seller, buy(car, 1200)), 13), \\ obl(ill(inform, ag_3, payer, ag_4, payee, pay(Price)), T_1, 13) : \{1200 \leq Price\}, \\ prh(ill(ask, ag_3, payer, X, adm, leave), T_2, 13), \\ oav(ag_3, credit, 3000), oav(car, price, 1200) \end{array} \right\}$$

Figure 4.3: Sample Institutional State

a seller agent – the seller agent ag_4 offers to sell a car for 1200 to buyer agent ag_3 who accepts the offer. The order among utterances is represented via time stamps (10 and 13 in the constructs above). In our example, agent ag_3 has agreed to buy the car so it is assigned an obligation to pay at least 1200 to agent ag_4 because a constraint restrict the values for *Price*, that is, the minimum value for the payment ; agent ag_3 is prohibited from asking the protocol administrator *adm* to leave. We employ a predicate *oav* (standing for *object-attribute-value*) to store attributes of our state: these concern the credit of agent ag_3 and the price of the car.

4.3 Providing Semantics to Deontic Notions

We now provide some examples on how we explicitly manage normative positions of agents in our language.When specifying a normative system we need to define relationships among deontic notions. Such relationships should capture the pragmatics of normative aspects – what exactly these concepts mean in terms of agents' behaviour. We do not want to be prescriptive in our discussion and we are aware that the sample rules we present can be given alternative formulations. Furthermore, we notice that when designing institutional rules, it is essential to consider the *combined* effect of the whole set of rules over the institutional states – these should be engineered in tandem.

We can confer different degrees of enforcement on MAS. We start by looking at those events that agents generate, *i.e.*, $att(E)$; these may become legal events, *i.e.*, $inst(E, T)$, if they are *permitted*, as specified by the following rule:

$$att(E) \ \texttt{\&} \ time(T) \ \texttt{\&} \ per(E, T, T_0) : C \ \texttt{\&} \ \texttt{sat}(C) \rightsquigarrow \texttt{add}(inst(E, T)) \qquad (4.4)$$

That is, permitted attempts at events become legal or institutionalised events.

Attempts and prohibitions can be related together by institutional rules of the form $att(E)$ & $prh(E, T_{inst}, T) \rightsquigarrow \texttt{del}(att(E)), sanction$ where *sanction*

stands for atomic formulae representing sanctions on the agent who generated a prohibited event. For instance, if the agent's credit is represented via $oav(Ag, credit, Value)$, the following rule applies a 10% fine on those agents who utter a prohibited illocution:

$$\begin{pmatrix} att(ill(P, A_1, R_1, A_2, R_2, M)) \ \texttt{\&} \ time(T) \ \texttt{\&} \\ prh(ill(P, A_1, R_1, A_2, R_2, M), T, T_0) : C \ \texttt{\&} \\ \texttt{sat}(C) \ \texttt{\&} \ oav(A_1, credit, C) \ \texttt{\&} \\ C2 = C - C/10 \end{pmatrix} \rightsquigarrow \begin{pmatrix} \texttt{del}(oav(A_1, credit, C)), \\ \texttt{add}(oav(A_1, credit, C2)) \end{pmatrix} \quad (4.5)$$

Another way of relating attempts, permissions and prohibitions is when a permission granted in general (*e.g.*, to all agents or to all agents adopting a role) is revoked for a particular agent (*e.g.*, due to a sanction). We can ensure that a permission has not been revoked via the rule:

$$\begin{pmatrix} att(E) \ \texttt{\&} \ time(T) \ \texttt{\&} \ per(E, T, T_0) : C \ \texttt{\&} \ \texttt{sat}(C) \ \texttt{\&} \\ \texttt{not}(prh(E, T, T_1) : C' \ \texttt{\&} \ \texttt{sat}(C')) \end{pmatrix} \rightsquigarrow \texttt{add}(inst(I, T)) \quad (4.6)$$

The rule above states that an utterance is accepted as legal whenever it is permitted and it is not the case that it is forbidden.

We can allow agents to do certain illegal actions (under harsher penalties if required):

$$\begin{array}{c} \left(\begin{array}{c} att(ill(inform, Ag_1, R, Ag_2, R', info(Ag_3, C))) \ \texttt{\&} \\ (Ag_1 \neq Ag_2) \ \texttt{\&} \ (Ag_1 \neq Ag_3) \ \texttt{\&} \ (Ag_2 \neq Ag_3) \ \texttt{\&} \ time(T) \end{array} \right) \\ \rightsquigarrow \\ \left(\begin{array}{c} \texttt{del}(att(ill(inform, Ag_1, R, Ag_2, R', info(Ag_3, C)))), \\ \texttt{add}(inst(ill(inform, Ag_1, R, Ag_2, R', info(Ag_3, C)), T)) \end{array} \right) \end{array} \quad (4.7)$$

The rule above states that if an agent Ag_1, enacting role R, attempts to reveal to Ag_2, enacting role R', (private) information C about agent Ag_3, and the three variables refer to different agents, then the attempt is accepted without taking into account if it is forbidden or not. In both cases (rules 4.6 and 4.7), we can punish agents that violate prohibitions as shown in rule 4.5.

The semantics of obligations also depends on which rules are part of the system. These rules should be selected taking into account the semantics of the obligatory events. For instance, when an agent fulfills its obligation to pay a certain amount of money, we remove that obligation as shown in rule 4.8. However, an obligation to be quiet in a given situation (notice that is equivalent to a prohibition to utter everything) may not need to be consumed each time

an agent is quiet and, therefore, no extra rule needs to be added.

$$\left(\begin{array}{c} att(ill(inform, Ag_1, payer, Ag_2, payee, pay(P))) \ \texttt{\&}\ time(T)\ \texttt{\&} \\ per(ill(inform, Ag_1, payer, Ag_2, payee, pay(P)), T, T_0) : C\ \texttt{\&}\ \texttt{sat}(C)\ \texttt{\&} \\ \texttt{not}(prh(ill(inform, Ag_1, payer, Ag_2, payee, pay(P)), T, T_1) : C'\ \texttt{\&}\ \texttt{sat}(C'))\ \texttt{\&} \\ obl(ill(inform, Ag_1, payer, Ag_2, payee, pay(P)), T, T_2) : C''\ \texttt{\&}\ \texttt{sat}(C'')\ \texttt{\&} \\ (Ag_1 \neq Ag_2) \end{array}\right)$$
$$\rightsquigarrow$$
$$\left(\begin{array}{c} \texttt{del}(att(ill(inform, Ag_1, payer, Ag_2, payee, pay(P)))), \\ \texttt{del}(obl(ill(inform, Ag_1, payer, Ag_2, payee, pay(P)), T, T_2)) \end{array}\right) \tag{4.8}$$

Let us consider now, that the agents are obliged to do actions before a certain deadline expressed with constraints. The designer of the MAS may choose to punish all the agents that does not fulfill a deadline with a fee of 20€.

Rule 4.9 states that if an obligation with deadline has not been fulfilled, i.e. there exists an obligation with a constraint associated to the time, the deadline has passed, i.e. current time is greater or equal to the deadline, and we have not yet applied a sanction for that particular obligation, then we apply a sanction:

$$\left(\begin{array}{c} obl(ill(inform, Ag_1, R, Ag_2, R', Action), T, T_0) : C\ \texttt{\&} \\ (T < D) \in C\ \texttt{\&}\ time(T2)\ \texttt{\&}\ (T2 \geq D)\ \texttt{\&} \\ \texttt{not}(sanction(obl(ill(inform, Ag_1, R, Ag_2, R', Action), T), (T < D))) \\ \texttt{\&}\ credit(Ag_1, C)\ \texttt{\&}\ credit(ei, C2)\ \texttt{\&}\ C3 = C - 20\ \texttt{\&}\ C4 = C2 + 20 \end{array}\right)$$
$$\rightsquigarrow$$
$$\left(\begin{array}{c} \texttt{del}(credit(Ag_1, C)), \texttt{add}(credit(Ag_1, C3)), \\ \texttt{del}(credit(ei, C2)), \texttt{add}(credit(ei, C4)), \\ \texttt{add}(sanction(obl(ill(inform, Ag_1, R, Ag_2, R', Action), T, T_0), (T < D))) \end{array}\right) \tag{4.9}$$

These examples show that our language can be used to build norm enforcement mechanisms.

4.4 Normative Conflict Resolution

We can also capture further relationships among normative aspects and establish policies to cope with conflicts. For instance, we need to specify how to cope with the situation when an illocution is simultaneously obliged and forbidden – this may occur when an obligation assigned to agents in general (or to any agents playing a role) is revoked for a particular subgroup of agents or an individual agent (for instance, due to a sanction). In this case, we can choose to ignore/override either the obligation or the prohibition. For instance, without writing any extra rule we override the obligation and ignore the attempt to fulfil the obligation. The rule below ignores the prohibition and transforms an attempt to utter the illocution I into its utterance:

$$att(E)\ \texttt{\&}\ time(T)\ \texttt{\&}\ obl(E, T, T_0)\ \texttt{\&}\ prh(E, T, T_1) \rightsquigarrow \texttt{add}(inst(E, T)) \tag{4.10}$$

A third possibility is to raise an exception via a term which can then be dealt with at the institutional level. The following rule could be used for this purpose:

$$att(E)\ \texttt{\&}\ time(T)\ \texttt{\&}\ obl(E, T, T_0)\ \texttt{\&}\ prh(E, T, T_1) \rightsquigarrow \texttt{add}(exc(E)) \quad (4.11)$$

These examples illustrate how we explicitly manage normative positions of agents in our language.

When adding constraints to predicates, some problems may rise:

1. constraints in a predicate may be erroneously specified as not satisfiable. For instance, a rule might try to add an obligation to an agent to pay more than 50€ and less than 20€.

2. constraints in related normative positions may be erroneously specified as not satisfiable. For instance, an agent may be obliged to pay more than 50€ but also forbidden to pay more than 20€.

A solution for the first problem is already included in the semantics of IRL, as we check in the RHS of the rules the satisfiability of constraints before adding a predicate. The second problem may be avoided by verification techniques at design time. However, these methods do not have good performance when the are too many constraints to check. In chapter 6, we will propose an algorithm to be applied at run-time to fix the problem of normative positions with unsatisfiable constraints.

4.4.1 Representing and Enacting Protocols via Institutional Rules

The purpose of this section is to represent and build a computational model of the dynamics of an interaction enactment based on protocols, that is, its execution with our rule-based language. Our model is based on scenes of EIs [Esteva, 2003] (see section 2.3.1) but our approach addresses any protocol specified via non-deterministic finite-state machines.

We shall represent protocols declaratively as logic programs, as described in [Vasconcelos et al., 2004]. Each edge connecting two states of a protocol will be denoted as the fact

$$edge(State, Event, NewState)$$

representing that if the control of the enactment of the protocol is in *State* and *Event* is uttered, then the control should move to *NewState*. Edges are compact descriptions of what can be performed, *i.e.*, the meaningful actions, and how the control of the enactment of the protocol (and by extension, of the MAS as a whole) should change as events are generated. Notice that although an agent may generate a meaningful event ($att(Event)$ and $edge(State, Event, NewState)$) in a given situation, it may also need to be permitted ($per(E, T_{inst}, T_0)$) and not prohibited ($\texttt{not}\ prh(E, T_{inst}, T_1)$) to do it so. By "meaningful" we mean that the event makes sense in the context of that protocol, that is, at a particular

point of the protocol, we specify via edges all possible events that agents may generate at any point. Of these, some will be permitted, as explained below.

Protocols are descriptions of what events may be performed and when they can be brought about in order to have a desired meaning. When permissions are combined with attempted events (*i.e*, $att(Event)$, as captured by formula 4.4 above) and approved utterances (*i.e*, $inst(Event, T)$) are combined with updates on the state of the enactment, then the protocol can be fully captured. In order to represent the control of the protocol enactment we use the term $ctr(State, TimeStamp)$, stored in the institutional state, which informs that at time $TimeStamp$ the protocol enacted is at $State$.

The dynamics of the control of the enactment can be captured generically as the following institutional rule:

$$\begin{pmatrix} state(W_i, T)\ \texttt{\&}\ time(T_2)\ \texttt{\&} \\ att(E)\ \texttt{\&}\ edge(W_i, E, W_j)\ \texttt{\&} \\ per(E, T_2, T_0) : C\ \texttt{\&}\ \texttt{sat}(C)\ \texttt{\&} \\ \texttt{not}(prh(E, T_2, T_1) : C'\ \texttt{\&}\ \texttt{sat}(C')) \end{pmatrix} \rightsquigarrow \begin{pmatrix} \texttt{del}(state(W_i, T)), \\ \texttt{add}(old_state(W_i, T)), \\ \texttt{add}(state(W_j, T_2)), \\ \texttt{add}(inst(E, T_2)) \end{pmatrix} \tag{4.12}$$

That is, if the control of the enactment of the protocol is at state W_i and event E has been performed, there is an edge connecting W_i with W_j labelled with that event, and the event is permitted and not prohibitted, then the state of the enactment at the next time will move to $state(W_j, T_2)$. We keep track of the time of previous states using the *old_state* predicate.

We notice that institutional rules are expressive enough to represent normative aspects as well as protocols (*i.e.*, interactions) and their enactment.

4.4.2 Example: The Dutch Auction Protocol

In this section, we illustrate the pragmatics of our norm-oriented language by specifying the auction protocol employed in the fish market described in [Noriega, 1997]. Following [Noriega, 1997], the fish market can be described as a place where several scenes [Esteva, 2003] take place simultaneously, at different locations, but with some causal connection. The principal scene is the auction itself, in which buyers bid for boxes of fish that are presented by an auctioneer who calls prices in descending order, following an *open cry, sudden death, downward bidding protocol*, a variation of the traditional Dutch auction protocol that proceeds as follows:

1. The auctioneer chooses a good out of a lot of goods that is sorted according to the order in which sellers deliver their goods to the sellers' admitter.

2. With a chosen good, the auctioneer opens a *bidding round* by quoting offers downward from the good's starting price, previously fixed by a sellers' admitter, as long as these price quotations are above a *reserve price* previously defined by the seller.

3. For each price the auctioneer calls, several situations might arise during the open round described below.

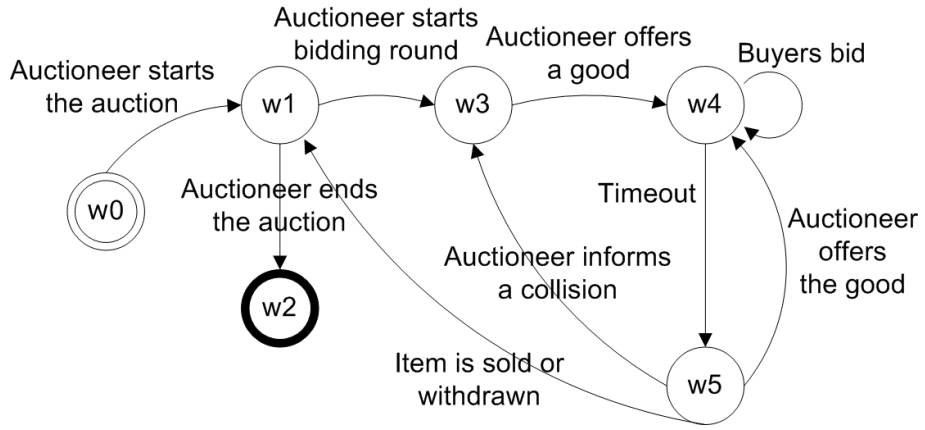

Figure 4.4: The Dutch Auction Protocol

4. The first three steps are repeated until there are no more goods left.

The situations arising in step 3 are:

Multiple bids – Several buyers submit their bids at the current price. In this case, a collision comes about, the good is not sold to any buyer, and the auctioneer restarts the round at a higher price;

One bid – Only one buyer submits a bid at the current price. The good is sold to this buyer whenever his credit can support his bid. Otherwise, the round is restarted by the auctioneer at a higher price, and the unsuccessful bidder is fined;

No bids – No buyer submits a bid at the current price. If the reserve price has not been reached yet, the auctioneer quotes a new price obtained by decreasing the current price according to the price step. Otherwise, the auctioneer declares the good as *withdrawn* and closes the round.

Proposed Solution

Figure 4.4 shows a finite state machine the protocol. Following section 4.4.1 the protocols are represented as a set of formula of the type $edge(W_i, E, W_j)$ and rule 4.12. The situations arising in step 3 are captured in rules 4.13 – 4.18. For formatting reasons, we will use α_i to denote atomic formulae:

Multiple bids – This rule obliges the auctioneer to inform the buyers, whenever a collision comes about, about the collision and obliges the auctioneer to restart the bidding round at a higher price (in this case, 120% of the collision price). Notice that X will hold all the utterances at scene *dutch* and state w_4 issued by buyer agents that bid for an item *It* at price P at time T_0 after the last offer. We obtain the last offers by checking that there

are no further offers whose time-stamps are greater than the time-stamp of the first one. If the number of illocutions in X is greater than one, the rule introduces the obligation above:

$$\left(\begin{array}{c} X = \{\ \alpha_0 \mid \alpha_1\ \texttt{\&}\ \texttt{not}(\alpha_2\ \texttt{\&}\ (T_2 > T_1))\ \texttt{\&}\ (T_0 > T_1)\ \} \\ \texttt{\&}\ (\texttt{size}(X) > 1)\ \texttt{\&}\ (P_m = P * 1.2)\ \texttt{\&}\ time(T_3) \end{array} \right) \rightsquigarrow (\ \texttt{add}(\alpha_3),\texttt{add}(\alpha_4)\ )$$

$$where \left\{ \begin{array}{l} \alpha_0 = inst(ill(inform, A_1, buyer, Au, auct, bid(It, P)), T_0) \\ \alpha_1 = inst(ill(inform, Au, auct, all, buyer, offer(It, P)), T_1) \\ \alpha_2 = inst(ill(inform, Au, auct, all, buyer, offer(It, P)), T_2) \\ \alpha_3 = obl(ill(inform, Au, auct, all, buyer, collision(It, P)), T_3, T_4) \\ \alpha_4 = obl(ill(inform, Au, auct, all, buyer, offer(It, P_2)), T_3, T_5) : \{P_2 > P_m\} \end{array} \right. \tag{4.13}$$

One bid/winner determination – If only one bid has occurred during the current bidding round and the credit of the bidding agent is greater than or equal to the price of the good in auction, the rule adds the obligation for the auctioneer to inform all the buyers about the sale:

$$\left(\begin{array}{l} X = \{\ \alpha_0 \mid \alpha_1\ \texttt{\&}\ \texttt{not}(\alpha_2\ \texttt{\&}\ (T_2 > T_1))\ \texttt{\&}\ (T_0 > T_1)\ \}\ \texttt{\&} \\ (\texttt{size}(X) = 1)\ \texttt{\&}\ oav(A_1, credit, C)\ \texttt{\&}\ (C \geq P)\ \texttt{\&}\ time(T_3) \end{array} \right) \rightsquigarrow (\ \texttt{add}(\alpha_3)\ )$$

$$where \left\{ \begin{array}{l} \alpha_0 = inst(ill(inform, A_1, buyer, Au, auct, bid(It, P)), T_0) \\ \alpha_1 = inst(ill(inform, Au, auct, all, buyer, offer(It, P)), T_1) \\ \alpha_2 = inst(ill(inform, Au, auct, all, buyer, offer(It, P)), T_2) \\ \alpha_3 = obl(ill(inform, Au, auct, all, buyer, sold(It, P, A_1)), T_4, T_3) \end{array} \right. \tag{4.14}$$

Prevention – We may prevent agents from issuing bids they cannot afford, that is, bids for which their credit is insufficient. The optional rule below states that if agent Ag's credit is less than P (the last offer the auctioneer called for item It), then agent Ag is prohibited to bid.

$$(\ \alpha_0\ \texttt{\&}\ \texttt{not}(\alpha_1\ \texttt{\&}\ (T2 > T))\ \texttt{\&}\ oav(Ag, credit, C)\ \texttt{\&}\ (C < P)\ \texttt{\&}\ time(T_3)\ ) \rightsquigarrow (\ \texttt{add}(\alpha_2)\ )$$

$$where \left\{ \begin{array}{l} \alpha_0 = inst(ill, inform, Au, auct, A, buyer, offer(It, P)), T) \\ \alpha_1 = inst(ill(inform, Au, auct, A, buyer, offer(It, P)), T_2) \\ \alpha_2 = prh(ill(inform, A, buyer, Au, auct, bid(It, P_2)), T_4, T_3)) \end{array} \right. \tag{4.15}$$

Punishment – Instead of preventing, we may punish those agents when issuing a winning bid they cannot pay for. More precisely, the rule punishes an agent A_1 by decreasing its credit of 10% of the value of the good being auctioned. The *oav* predicate on the LHS of the rule represents the current credit of the offending agent. The rule also adds an obligation for the auctioneer to restart the bidding round and the constraint that the new offer should be greater than 120% of the old price.

$$\left(\begin{array}{c} X = \left\{ \begin{array}{l} \alpha_0 \\ \mid \end{array} \begin{array}{l} \alpha_1\ \texttt{\&}\ (T_0 > T_1)\ \texttt{\&} \\ \texttt{not}(\alpha_2\ \texttt{\&}\ (T_2 > T_1)) \end{array} \right\}\ \texttt{\&} \\ oav(A_1, credit, C)\ \texttt{\&} \\ (\texttt{size}(X) = 1)\ \texttt{\&}\ (C < P)\ \texttt{\&} \\ C2 = C - P * 0.1\ \texttt{\&}\ P_o = P * 1.2 \end{array} \right) \rightsquigarrow \left(\begin{array}{c} \texttt{del}(oav(A_1, credit, C)), \\ \texttt{add}(oav(A_1, credit, C2)), \\ \texttt{add}(\alpha_3) \end{array} \right)$$

$$where \left\{ \begin{array}{l} \alpha_0 = inst(ill(inform, A_1, buyer, Au, auct, bid(It, P)), T_0) \\ \alpha_1 = inst(ill(inform, Au, auct, all, buyer, offer(It, P)), T_1), \\ \alpha_2 = inst(ill(inform, Au, auct, all, buyer, offer(It, P)), T_2) \\ \alpha_3 = obl(ill(inform, Au, auct, all, buyer, offer(It, P_o)), T_3) \end{array} \right. \tag{4.16}$$

No bids/New Price – We must check if there were no bids and if the next price is greater than the reserve price. If so, we must add an obligation for the auctioneer to start a new bidding round. Rule 4.17 checks that the current scene state is w_5, the last offer occurred before w_5 and whether the new price is greater than reserve price. If so, the rule adds the obligation for the auctioneer to offer the item at a lower price. By retrieving the last offer we gather the last offer price. By checking the *oav* predicates we gather the values of the reserve price and the decrement rate for item *It*.

$$\left(\begin{array}{c} ctr(dutch, w_5, T_n)\ \texttt{\&}\ \alpha_0\ \texttt{\&} \\ \texttt{not}(\alpha_1\ \texttt{\&}\ (T_2 > T))\ \texttt{\&}\ (T_n > T)\ \texttt{\&} \\ oav(IT, reservation_price, RP)\ \texttt{\&} \\ oav(IT, decrement_rate, DR)\ \texttt{\&} \\ (RP < (P - DR))\ \texttt{\&}\ (P_2 = P - DR)\ \texttt{\&}\ time(T_3) \end{array}\right) \rightsquigarrow (\ \texttt{add}(\alpha_2)\ ) \tag{4.17}$$

$$where \left\{\begin{array}{l} \alpha_0 = instill(inform, Au, auct, all, buyer, offer(IT, P)), T) \\ \alpha_1 = inst(ill(inform, Au, auct, all, buyer, offer(IT, P)), T_2) \\ \alpha_2 = obl(ill(inform, Au, auct, all, buyer, offer(IT, P2)), T_4, T_3) \end{array}\right.$$

No bids/withdrawal – We must check if there were no bids and the next price is less than the reserve price; if so we add the obligation for the auctioneer to withdraw the item. Rule 4.18 checks that the current institutional state is w_5, the last offer occurred before w_5 and whether the new offer price is greater than reserve price. If the LHS holds, the rule fires to add the obligation for the auctioneer to withdraw the item. By checking the last offer we gather the last offer price. By checking the *oav* predicates we gather the values of the reserve price and the decrement rate for the price of item *It*:

$$\left(\begin{array}{c} ctr(dutch, w_5, T_n)\ \texttt{\&}\ \alpha_0\ \texttt{\&} \\ \texttt{not}(\alpha_1\ \texttt{\&}\ (T_2 > T))\ \texttt{\&}\ (T_n > T)\ \texttt{\&} \\ oav(It, reservation_price, RP)\ \texttt{\&} \\ oav(It, decrement_rate, DR)\ \texttt{\&} \\ (RP \geq (P - DR))\ \texttt{\&}\ time(T_3) \end{array}\right) \rightsquigarrow (\ \texttt{add}(\alpha_2)\ ) \tag{4.18}$$

$$where \left\{\begin{array}{l} \alpha_0 = inst(ill(inform, Au, auct, all, buyer, offer(It, P)), T) \\ \alpha_1 = inst(ill(inform, Au, auct, all, buyer, offer(It, P)), T_2) \\ \alpha_2 = obl(ill(inform, Au, auct, all, buyer, withdrawn(It)), T_4, T_3) \end{array}\right.$$

4.5 Conclusions

In this chapter, to continue answering our first two research questions, we have introduced a language for the explicit management of normative positions bounded by arithmetical constraints and specifying the behaviour of agents in MASs. The classical model of EIs proposed in [Esteva, 2003] is strict in the sense that only meaningful speech acts are accepted in the protocols without checking if they are permitted or not. We propose a language to extend the notion of EI by implementing its scenes and providing them with several flavours of deontic notions.

The main advantage of using our language, instead of standard production systems (such as Jess, used in Chapter 4), to specify and monitor the normative position of the agents conforming a MAS is to be able to avoid forward-chaining

mechanisms to easily implement one-time actions and sanctions, and the inclusion of constraint solving techniques in the semantics to handle constrained predicates, *i.e.* to manage constraints and to check how these constraints affect the predicates they constrain. We have obtained a language to express, manage, check fulfilment and/or sanction unfulfilled normative positions, i.e. obligations, permissions and prohibitions, that are bounded with constraints.

Thus, the language presented in this chapter is useful to predict a future state of affairs with an initial state and a sequence of sets of events that occur by modifying the intermediate states of affairs until we reach the final one. The limitations of the language are determined by the rule engine. These limitations include the inability to plan, i.e. determine the sequence of sets of events that must occur in order to reach a given state of affairs from a given initial state, or post-dicting, i.e. determine the previously unknown facts in a partial initial state given a final state and the sequence of sets of events that have occurred. However, the goal of the language is to regulate a MAS and keep track of its evolution by prediction. Post-diction and planning would be interesting for a language that an agent could use for deciding which action to perform but this is not the aim of this thesis.

We envisage two typical ways of using our language: i) directly, to supplement a MAS with interaction protocols and norms or declaratively implement scenes of EIs or ii) specifying norms with a language like the one presented in chapter 4 and then using a compiler to translate it into IRL to execute it.

Although the proposals of this chapter answers the first two research questions posed in Chapter 1, we noticed that we can provide more expressiveness to candidate languages by exploiting more the computational model presented in this chapter for the specification of simultaneous speech acts that can be permitted, prohibited and forbidden. Furthermore, we can improve the semantics of our language by including some enforcement actions, that we call institutional actions, such as ignoring, expecting or forcing certain simultaneous speech acts or preventing a given state of affairs. Thus, we will include this in the proposal of the next chapter.

Chapter 5

Regulating Concurrency

In this chapter we propose a language with different types of rules to specify norms (with temporal aspects and arithmetical constraints) over agents' simultaneous speech acts and to specify preventive and corrective actions that the system has to perform. We also propose ignoring speech acts and preventing states of affairs as preventive actions and forcing speech acts and sanctioning as corrective actions. Finally, we provide an interpreter for the language shown, in its entirety, in Appendix B.

As shown in chapter 2, in the literature we find that almost all normative languages are based on deontic logics establishing which actions are permitted, forbidden or obligatory. However, deontic logics does not establish the semantics of these modalities with respect to a computational system. For instance, when an action is claimed to be forbidden, does it means that it is prevented from happening, or that the agents that bring it about must be sanctioned or that the effects of that action are just ignored?

Thus, in this chapter we continue improving the answer for the first two research questions we posed in chapter 1: how to specify norms (and make them operational) to regulate a multi-agent activity. In this chapter, we propose $\mathcal{I}$, a language that reshapes our IRL rules into *Event-Condition-Action* (ECA) rules that perform the given actions when the given events occur and the given conditions hold. The performance of actions add or remove atomic formulae thus triggering another type of rule: the *if-rules*, that are standard production rules. Above these two types of rules we place new types of rules: *ignore-rules*, that ignore a set of simultaneous events, *force-rules*, that generate a set of events on the occurrence of a given set of events satisfying certain conditions, and *prevent-rules*, that ignore the execution of *ECA* or *if-then* rules if certain formulae hold in the current state and another given set of formulae hold in the next state.

The main contributions of $\mathcal{I}$ is the management of sets of events that occur simultaneously and the distinction between norms that can be violated or not. For instance, let us consider the case when an agent wins a given good in an auction. We envisage two options for the payment in that scenario: *1)* expect the agent to generate the event of the payment and sanction the agent if the event

is not generated before a given deadline; or *2)* if the institution has control on the agent's balance, automatically generate an event of payment as if the agent would have generated it. In fact, an obligation (to perform an event) that may be violated is represented as the expectation of the attempts to perform it. However, the enforcement of an obligation (to perform a set of events) that may not be violated is carried out by the system by taking these events as having been performed even they have not. We denote such enforcement as forcing events.

5.1 $\mathcal{I}$: A Language for Institutions

A problem that we spotted during the development of the previous language is that if two agents simultaneously perform actions, *i.e.* gathered during the same round, that modifies the same predicate, *e.g.* the state of the scene, the predicate is duplicated with the values of each institutionalised action. A partial solution would be modifying all the rules to check if there is only one attempt institutionalisable prior to institutionalise it. However, this rule should check for all the attempts that comply with any LHS of the rules that in their RHS institutionalise an event. Writing this rule is a hard task to leave it to the programmers of electronic institutions. We need then, to include in the semantics a mechanism to avoid certain state of affairs to come about.

In this section we introduce a rule language for the regulation and management of concurrent events generated by a population of agents. Our rule-based language allows us to represent norms and changes in an elegant way.

As in previous chapter, the building blocks of our language are first-order terms and implicitly, universally quantified atomic formulae without free variables and we adopt Prolog's convention using strings starting with a capital letter to represent variables and strings starting with a small letter to represent constants.

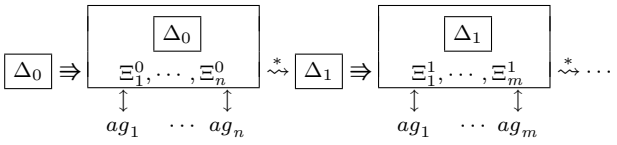

Figure 5.1: Semantics as a Sequence of Δ's

Figure 5.1 depicts again the computational model we continue using in this chapter. Recall from Figure 4.1 that an initial state of affairs Δ_0 (possibly empty) is offered (represented by "$\Rightarrow$") to a set of agents $(ag_1, \cdots, ag_n)$. These agents can add their utterances $(\Xi_1^0, \cdots, \Xi_n^0)$ to the state of affairs (via "$\updownarrow$"). Ξ_i^t is the (possibly empty) set of illocutions added by agent i at state of affairs Δ_t. After an established amount of time, we perform an exhaustive application of

rules (denoted by "$\overset{*}{\rightsquigarrow}$") to the modified state, yielding a new state of affairs Δ_1. This new state will, on its turn, be offered to the agents for them to add their utterances, and the same process will go on.

One goal of the $\mathcal{I}$ language is to specify the effects of concurrent events and this is achieved with Event-Condition-Action (ECA) rules. Intuitively, an ECA-rule means that whenever the events occur and the conditions hold then the actions are applied. These actions consist of the addition and removal of atomic formulae from the state of affairs. ECA-rules are checked in parallel and they are executed only once without chaining.

If-rules are similar to rules in standard production systems, if the conditions hold then the actions are applied. They are implemented with a forward chaining mechanism: they are executed sequentially until no new formula is added or removed.

Ignore-rules are used for ignoring events when the conditions hold in order to avoid unwanted behaviour. Similarly, prevent-rules are used for preventing some conditions to hold in the given situations. In order to prevent unwanted states, events causing such unwanted states are ignored. Force-rules generate events and execute actions as consequence of other events and conditions.

Sanctions over unwanted events can be carried out with ECA-rules. For instance, we can decrease the credit of one agent by 10 if she generates certain event.

ECA-Rule	::=	**on** *set_of_events* **if** *conditions* **do** *actions*
if-Rule	::=	**if** *conditions* **do** *actions*
ignore-Rule	::=	**ignore** *set_of_events* **if** *conditions*
prevent-Rule	::=	**prevent** *conditions* **if** *conditions*
force-Rule	::=	**force** *set_of_events* **on** *set_of_events*
		if *conditions* **do** *actions*
set_of_events	::=	*events* | $\emptyset$
events	::=	*atomic_formula*, *events* | *atomic_formula*
conditions	::=	*conditions* `&` *conditions* | `not`(*conditions*)
	|	`sat`(*set_of_constraints*) | *constr_formula*
	|	`seteq`(*set_of_constraints*, *set_of_constraints*)
	|	*constraint* $\in$ *set_of_constraints*
	|	`time`(*number*) | `true`
constr_formula	::=	*atomic_formulae*
	|	*atomic_formulae* : *set_of_constraints*
actions	::=	*action* `,` *actions* | *action*
action	::=	`add`(*constr_formula*) | `del`(*constr_formula*)

Figure 5.2: Grammar for $\mathcal{I}$

Figure 5.2 shows the grammar for $\mathcal{I}$, *i.e.* the syntax of the five type of rules we propose: ECA-rules, if-rules, ignore-rules, prevent-rules and force-rules.

ECA rules specify the effect of a set of events, *i.e.* a set of atomic formulae, if the conditions hold. This effect is specified by means of a sequence of actions

namely addition and removal of constrained formulae. A constrained formulae is an atomic formula that may be followed by a set of arithmetical constraints using the syntax presented in Def. 4.1.3. Furthermore, by *conditions* we mean one or more possibly negated conditions. Then, a condition may be a constrained formula, the `sat` predicate that checks that a set of constraints are satisfiable, the `seteq` predicate that checks if two sets are equal, the `time`predicate that checks current time or the `true` constant that always hold.

If-rules specify the logical consequence if the conditions hold by means of a sequence of actions. Ignore-rules specify the set of events that should be ignored if the conditions hold. Similarly, prevent-rules specify the conditions that should not hold if some conditions hold. Finally, force-rules specify a set of new events that are generated on the occurrence of a set of events and the satisfaction of a sequence of conditions. Furthermore, it also specifies a sequence of actions to perform if the rule is triggered.

We add an extra kind of rules, called expectation-rules, that generate and remove expectations of events. If the expectation fails to be fulfilled then some sanctioning or corrective actions are performed.

$$\begin{aligned} \textit{expectation-Rule} \quad ::= \quad & \textbf{expected } \textit{event } \textbf{on } \textit{set_of_events } \textbf{if } \textit{conditions} \\ & \textbf{fulfilled-if } \textit{conditions}' \textbf{ violated-if } \textit{conditions}'' \\ & \textbf{sanction-do } \textit{actions} \end{aligned}$$

Each expectation rule can be translated into the following rules:

$$\textbf{on } \textit{set_of_events } \textbf{if } \textit{conditions } \textbf{do } \texttt{add}(\textit{exp}(\textit{event})) \tag{5.1}$$

$$\textbf{if } \textit{exp}(\textit{event}) \wedge \textit{conditions}' \textbf{ do } \texttt{del}(\textit{exp}(\textit{event})) \tag{5.2}$$

$$\textbf{if } \textit{exp}(\textit{event}) \wedge \textit{conditions}'' \textbf{ do } \texttt{del}(\textit{exp}(\textit{event})), \textit{actions} \tag{5.3}$$

Rule 5.1 and 5.2 respectively adds and removes an expectation whenever the events have occurred and the conditions hold. Rule 5.3 cancels the unfulfilled expectation and sanctions an agent for the unfulfilled expectation by executing the given *actions* whenever some *conditions* hold.

5.1.1 Semantics

Instead of basing the $\mathcal{I}$ language on the standard deontic notions, two types of prohibitions and two types of obligations are included. In our language, ECA-rules determine what is possible to perform, i.e. they establish the effects (including sanctions) in the institution after performing certain (possibly concurrent) events. ECA-rules can be seen as conditional count-as rules: the given events count as the execution of the actions in the ECA-rule if the conditions hold and the event is not explicitly prohibited. As for the notion of permission, all the events are permitted if not explicitly prohibited. The notion of an event being prohibited may be expressed depending on whether that event has to be ignored or not. If not otherwise expressed, events are not ignored. Likewise, the notion of a state being prohibited may be specified depending on whether that

state has to be prevented or not. By default, states are not prevented. Obligations are differentiated in two types: expectations, which an agent may not fulfill, and forced (or obligatory) events, which the system takes as institutional events even they are not actually performed by the agents.

Each set of ECA-rules generates a labelled transition system $\langle \mathcal{S}, \mathcal{E}, \mathcal{R} \rangle$ where $\mathcal{S}$ is a set of states, each state in $\mathcal{S}$ is a set of atomic formulae, $\mathcal{E}$ is a set of events, and $\mathcal{R}$ is a $\mathcal{S} \times 2^{\mathcal{E}} \times \mathcal{S}$ relationship indicating that whenever a set of events occur in the former state, then there is a transition to the subsequent state.

Ignore-rules avoid executing any transition that contains in its labelling the events that appear in any ignore-rule. For instance, having a rule **ignore** α_1 **if** *true* would avoid to execute the transitions labelled as $\{\alpha_1\}$, $\{\alpha_1, \alpha_2\}$ and $\{\alpha_1, \alpha_2, \alpha_3\}$. However, having a rule **ignore** α_1, α_2 **if** *true* would avoid executing $\{\alpha_1, \alpha_2\}$ and $\{\alpha_1, \alpha_2, \alpha_3\}$ but not $\{\alpha_1\}$.

Prevent-rules ignore all the actions in an ECA-rule if it brings the given formulae about. For example, suppose that we have

prevent q_1 **if** *true*

along with ECA-rules 5.4, 5.5 and 5.6 below. After the occurrence of events α_1 and α_2 and since q_1 is an effect of event α_2, all the actions in ECA-rule 5.5 would be ignored obtaining a new state where p and r hold but neither q_1 nor q_2.

$$\textbf{on } \alpha_1 \textbf{ if } \textit{true} \textbf{ do } \texttt{add}(p) \tag{5.4}$$

$$\textbf{on } \alpha_2 \textbf{ if } \textit{true} \textbf{ do } \texttt{add}(q_1)\texttt{,add}(q_2) \tag{5.5}$$

$$\textbf{on } \alpha_1, \alpha_2 \textbf{ if } \textit{true} \textbf{ do } \texttt{add}(r) \tag{5.6}$$

Force-rules generate events during the execution of the transition system. However, the effects of such events are still specified by ECA-rules and subject to prevent and ignore-rules.

5.1.2 Operational Semantics

We now define the semantics of the conditions, that is, when a condition holds:

Definition 5.1.1. Relation $\mathbf{s}_l(\Delta, C, \sigma)$ holds between state Δ, a condition C in an **if** clause and a substitution σ depending on the format of the condition:

1. $\mathbf{s}_l(\Delta, C \ \texttt{\&} \ C', \sigma)$ holds iff $\mathbf{s}_l(\Delta, C, \sigma')$ and $\mathbf{s}_l(\Delta, C' \cdot \sigma', \sigma'')$ hold and $\sigma = \sigma' \cup \sigma''$.
2. $\mathbf{s}_l(\Delta, \texttt{not}(C), \sigma)$ holds iff $\mathbf{s}_l(\Delta, C, \sigma)$ does not hold.
3. $\mathbf{s}_l(\Delta, \texttt{seteq}(L, L2), \sigma)$ holds iff $L \subseteq L2$, $L2 \subseteq L$ and $\mid L \mid = \mid L2 \mid$.
4. $\mathbf{s}_l(\Delta, \texttt{sat}(\mathit{constraints}), \sigma)$ holds iff $satisfiable(\mathit{constraints} \cdot \sigma)$ hold.
5. $\mathbf{s}_l(\Delta, \gamma \in \Gamma, \sigma)$ holds iff $(\gamma \cdot \sigma) \in (\Gamma \cdot \sigma)$.
6. $\mathbf{s}_l(\Delta, \texttt{time}(T), \sigma)$ holds iff current time is T.

7. $\mathbf{s}_l(\Delta, \texttt{true}, \sigma)$ always holds.

8. $\mathbf{s}_l(\Delta, \mathit{constr_formula}, \sigma)$ holds iff $\mathit{constr_formula} \cdot \sigma \in \Delta$.

Case 1 depicts the semantics of atomic formulae and how their individual substitutions are combined to provide the semantics for a conjunction. Case 2 introduces negation by failure. Case 3 compares if two lists have the same elements possibly in different order. Case 4 checks if a set of constraints is satisfiable. Case 5 checks if a constraint belongs to a set of constraints. Case 6 checks if T is current time. Case 7 gives semantics to the keyword `true`. Case 8 holds when an possibly constrained, atomic formulae *constr_formula* is part of the state of affairs.

We now define the semantics of the actions of a rule:

Definition 5.1.2. Relation $\mathbf{s}_r(\Delta, A, \Delta')$ mapping a state Δ, the action section of a rule and a new state Δ' is defined as:

1. $\mathbf{s}_r(\Delta, (A\,\texttt{,}\,As), \Delta')$ holds iff both $\mathbf{s}_r(\Delta, A, \Delta_1)$ and $\mathbf{s}_r(\Delta_1, As, \Delta')$ hold.

2. $\mathbf{s}_r(\Delta, \texttt{add}(\mathit{constr_formula}), \Delta')$ holds iff

   (a) $\mathit{constr_formula} \notin \Delta$ and $\Delta' = \Delta \cup \{\mathit{constr_formula}\}$ or;

   (b) $\Delta' = \Delta$.

3. $\mathbf{s}_r(\Delta, \texttt{del}(\mathit{constr_formula}), \Delta')$ holds iff

   (a) $\mathit{constr_formula} \in \Delta$ and $\Delta' = \Delta \setminus \{\mathit{constr_formula}\}$ or;

   (b) $\Delta' = \Delta$.

Case 1 decomposes a conjunction and builds the new state by merging the partial states of each update. Case 2 and 3 cater respectively for the insertion and removal of atomic formulae α.

We now define relation check_{prv} that checks if there is no prevent-rule that has been violated, i.e., it is not the case that all the conditions of any prevent-rule hold in the state of affairs Δ'. It checks whether Δ' contain all the conditions of each prevent-rule or not, if Δ also contain the given conditions.

Definition 5.1.3. Relation $\mathit{check}_{prv}(\Delta, \Delta', \mathit{PrvRules})$ mapping Δ, the state before applying updates, Δ', the state after applying updates, and a sequence *PrvRules* of prevent-rules, holds iff an empty set is the largest set of conditions C such that prevent-rule $p =$ **prevent** C **if** C', $p \in \mathit{PrvRules}$, $\mathbf{s}_l(\Delta, C')$ and $\mathbf{s}_l(\Delta', C)$ hold.

Definition 5.1.4. Relation $\mathit{fire}(\Delta, \mathit{PrvRules}, \textbf{if } C \textbf{ do } A, \Delta')$ mapping a state Δ, a sequence *PrvRules* of prevent-rules, an if-rule and a new state Δ' holds iff $\mathit{fired}(C, A)$ starts to hold, $\mathbf{s}_r(\Delta, A, \Delta')$ and $\mathit{check}_{prv}(\Delta, \Delta', \mathit{PrvRules})$ hold.

Relation *can_fire* checks whether the conditions of a given if-rule hold and the rule after applying substitution σ has not been already fired.

Definition 5.1.5. Relation *can_fire*$(\Delta, \textbf{if } C \textbf{ do } A, \sigma)$ mapping a state Δ an if-rule and a substitution σ holds iff $\mathbf{s}_l(\Delta, C, \sigma)$ holds and *fired*$(C \cdot \sigma, A \cdot \sigma)$ does not hold.

Relation *resolve* determines the rule that will be fired by selecting the first rule in the list.

Definition 5.1.6. Relation *resolve*(*RuleList*, *SelectedRuleList*) mapping a list of if-rules and a selected if-rule list holds iff

1. *RuleList* $= \langle\rangle$ and *SelectedRuleList* $= \langle\rangle$; or
2. *RuleList* $= \langle r_1, \cdots, r_n\rangle$ and *SelectedRuleList* $= \langle r_1\rangle$.

Relation *select_rule* determines the rule that will be fired by selecting all the rules that can fire and resolving the conflict with relation *resolve*.

Definition 5.1.7. Relation *select_rule*$(\Delta,$ *IfRulesList*, *SelectedRuleList*) mapping a state of affairs Δ, a list of if-rules and a selected if-rule list holds iff Rs is the largest set of rules $R \in Rs$, $Rs \subseteq$ *IfRulesList* such that *can_fire*(Δ, R, σ); *resolve*(Rs, SR) hold and *SelectedRuleList* $= SR \cdot \sigma$.

Relation $\mathbf{s}_{if}$ determines the new state of affairs after applying a set of if-rules to a initial state of affairs taking into account a set of prevent-rules.

Definition 5.1.8. Relation $\mathbf{s}_{if}(\Delta,$ *IfRules*, *PrvRules*, $\Delta')$ mapping a state of affairs Δ, a list of if-rules, a list of prevent-rules and a new state of affairs holds iff

1. *select_rule*$(\Delta,$ *IfRules*, $R)$ hold, $R \neq \langle\rangle$, *fire*$(\Delta,$ *PrvRules*, $R, \Delta'')$ and $\mathbf{s}_{if}(\Delta'',$ *IfRules*, *PrvRules*, $\Delta')$ hold; or
2. *select_rule*$(\Delta,$ *IfRules*, $R)$ hold, $R = \langle\rangle$; or
3. $\mathbf{s}_{if}(\Delta,$ *IfRules*, *PrvRules*, $\Delta')$ hold.

Relation *ignored* determines that a set of events which occurred have to be ignored taking into account a list of ignore-rules.

Definition 5.1.9. Relation *ignored*$(\Delta, \Xi,$ $E,$ *IgnRules*) mapping a state of affairs Δ, a list Ξ of events that occurred, a list of events in a ECA-rule and a list of ignore-rules holds iff $i = \textbf{ignore } E' \textbf{ if } C$, $i \in$ *IgnRules*, $E' \subseteq \Xi$, $E \cap E' \neq \emptyset$ and $\mathbf{s}_l(\Delta, C)$ holds.

Relation $\mathbf{s}'_r$ uses $\mathbf{s}_r$ first and then $\mathbf{s}_{if}$ in order to activate the forward chaining.

Definition 5.1.10. Relation $\mathbf{s}'_r(\Delta,$ *IfRules*, *PrvRules*, *ActionList*, $\Delta')$ mapping a state of affairs Δ, a list of if-rules, a list of prevent-rules, a list of actions and a new state of affairs holds iff any of the conditions below hold:

1. *ActionList* $= \langle\rangle$ and $\Delta' = \Delta$; or

2. $ActionList = \langle a_1, \cdots, a_n \rangle$, $\mathbf{s}_r(\Delta, a_1, \Delta'')$, $check_{prv}(\Delta, \Delta'', PrvRules)$, $\mathbf{s}_{if}($ $\Delta'', IfRules, PrvRules, \Delta''')$ and $\mathbf{s}'_r(\Delta''', IfRules, PrvRules, \langle a_2, \cdots, a_n \rangle, \Delta')$ hold; or
3. $\mathbf{s}'_r(\Delta, IfRules, PrvRules, \langle a_2, \cdots, a_n \rangle, \Delta')$.

Relation $\mathbf{s}_{eca}$ calculates the new state of affairs Δ' from an initial state Δ and a set Ξ of events that occurred applying a list of ECA-rules, if-rules, ignore-rules and prevent-rules.

Definition 5.1.11. Relation $\mathbf{s}_{eca}(\Delta, \Xi, ECARules, IfRules, IgnRules, PrvRules, \Delta')$ mapping a state of affairs Δ, a list Ξ of events that occurred, a list of ECA-rules, a list of if-rules, a list of ignore-rules, a list of prevent-rules, and a new state of affairs holds iff:

- As is the largest set of actions $A' = A \cdot \sigma$ in an ECA-rule $r =$ **on** E **if** C **do** A such that:
  - $r \in ECARules$, $E \cdot \sigma' \subseteq \Xi$, $\mathbf{s}_l(\Delta, C, \sigma'')$ hold,
  - $ignored(\Delta, \Xi, E, IgnRules)$ does not hold and
  - $\sigma = \sigma' \cup \sigma''$; and
- $\mathbf{s}'_r(\Delta, IfRules, PrvRules, As, \Delta')$ hold.

Relation $\mathbf{s}_{force}$ calculates the new state of affairs Δ' and the new set Ξ' of occurred events from an initial state Δ and a set Ξ of events that occurred applying a list of if-rules, ignore-rules, prevent-rules and force-rules.

Definition 5.1.12. Relation $\mathbf{s}_{force}(\Delta, \Xi, IfRules, IgnRules, PrvRules, FrcRules, \Xi', \Delta')$ mapping a state of affairs Δ, a list Ξ of events that occurred, a list of if-rules, a list of ignore-rules, a list of prevent-rules, a list of force-rules, a new list of events that occured and a new state of affairs holds iff:

- EAs is the largest set of tuples $\langle FE \cdot \sigma, A \cdot \sigma \rangle$ of forced events and actions in a force rule $fr =$ **force** FE **on** E **if** C **do** A such that
  - $fr \in FrcRules$, $E \cdot \sigma' \subseteq \Xi$, $\mathbf{s}_l(\Delta, C, \sigma'')$ holds,
  - $ignored(\Delta, \Xi, E, IgnRules)$ does not hold and
  - $\sigma = \sigma' \cup \sigma''$;
- Es is the largest set of forced events Ev such that $\langle Ev, A \rangle \in EAs$;
- $\Xi' = \Xi \cup Es$;
- As is the largest set of actions A such that $\langle Ev, A \rangle \in EAs$; and
- $\mathbf{s}'_r(\Delta, IfRules, PrvRules, As, \Delta')$ holds.

Relation $\mathbf{s}^*$ calculates the new state of affairs Δ' from an initial state Δ and a set Ξ of events that occurred applying a list of ECA-rules, if-rules, ignore-rules, prevent-rules and force-rules.

Definition 5.1.13. Relation $\mathbf{s}^*(\Delta, \Xi, \mathit{ECARules}, \mathit{IfRules}, \mathit{IgnRules}, \mathit{PrvRules}, \mathit{FrcRules}, \Delta')$ mapping a state of affairs Δ, a list Ξ of events that occurred, a list of ECA-rules, a list of if-rules, a list of ignore-rules, a list of prevent-rules, a list of force-rules and a new state of affairs holds iff:

- Cs is the largest set of conditions C such that $\mathit{fired}(C, A)$ stop holding;
- $\mathit{fired}(\mathit{false}, \mathit{false})$ starts to hold,
- $\mathbf{s}_{if}(\Delta, \mathit{IfRules}, \mathit{PrvRules}, \Delta'')$,
- $\mathbf{s}_{force}(\Delta'', \Xi, \mathit{IfRules}, \mathit{IgnRules}, \mathit{PrvRules}, \mathit{FrcRules}, \Xi', \Delta''')$ and
- $\mathbf{s}_{eca}(\Delta''', \Xi', \mathit{ECARules}, \mathit{IfRules}, \mathit{IgnRules}, \mathit{PrvRules}, \Delta')$ hold.

5.1.3 Interpreter

In this section we introduce the main predicates we use in the interpreter of language $\mathcal{I}$ that we entirely present in Appendix B.

Figure 5.3 shows the top level predicate of our interpreter. Mirroring Def. 5.1.13, `s_star` calculates a new state of affairs triggering first if-rules to calculate logical consequences in initial state `Delta`; then it applies force-rules to create new forced events; and finally it executes ECA-rules to calculate the causal consequence of the initial events plus the generated ones. To apply force-rules and ECA rules, a checking of ignore-rules is performed, *i.e.* checking if any of the triggering events is ignored. After each if-rule, force-rule and ECA rule it checks if any prevent-rule is applicable to reject the triggering of the rule.

Recall from Defs. 5.1.7 and 5.1.6 that we chose if-rules to be triggered in declaration order. Similarly, by using `findall/3` Prolog predicate, we also assume the rest of rules to be triggered in declaration order but no rule resolution mechanism is provided in our prototype. We acknowledge that ordering of rules may be a desirable feature in production systems so we envisage a more expressive although more complex definitions of rule triggering including the feature mentioned above.

In Defs. 5.1.10 and 5.1.2 we introduced how actions in rules change the initial state by adding and removing possibly constrained atomic formulae. We use forward chaining for if-rules to calculate logical consequence of facts, that is, we execute rules until no new rule can be executed. In order to do not repeat the execution of the same rule with the same parameters we record them during one exhaustive application of rules.

Thus, starting from a inital state `Delta`, *i.e.* a list of atomic formulae, a list of events `Events`, and a list of each type of rule, the predicate calculates a new state `NewDelta`.

Then, predicate `s_star` calculates a new state performing the following procedure: it resets the annotations on rules previously fired, *i.e.* executed; it then calculates a partial state `Delta2` by applying if-rules; afterwards it modifies the list of events `Events`, and the partial state `Delta2` with the application of force rules obtaining a new list of events `NewEvents` and a new partial state `Delta3`;

```
s_star(Delta,Events,ECARules,IfRules,IgnRules,PrvRules,FrcRules,
  NewDelta):-
    reset,
    s_if(Delta,IfRules,PrvRules,Delta2),
    s_force(Delta2,Events,IfRules,IgnRules,PrvRules,FrcRules,
      NewEvents, Delta3),
    s_eca(Delta3,NewEvents,ECARules,IfRules,IgnRules,PrvRules,
      NewDelta).
```

Figure 5.3: The `s_star` predicate

finally it applies ECA-rules on the partial state `Delta3` using the new list of events `NewEvents` and obtaining the final state `NewDelta`.

As figures 5.4 and 5.5 show, the application of if-rules, mirroring Def. 5.1.8, consist on the repetition of selecting the first rule that can be fired and apply it until no new rule can be selected. To execute or fire a rule consist on annotating that the rule has been fired with the given parameters, calculating the new state after applying the actions of the rule and checking if no prevent-rule has to avoid the result.

```
s_if(Delta,IfRules,PrvRules,NewDelta):-
    select_rule(Delta,IfRules,R),R\=[],
    fire(Delta,PrvRules,R,TmpDelta),
    s_if(TmpDelta,IfRules,PrvRules,NewDelta).
s_if(Delta,IfRules,_,Delta):-
    select_rule(Delta,IfRules,[]).
s_if(Delta,IfRules,PrvRules,NewDelta):-
    s_if(Delta,IfRules,PrvRules,NewDelta).
```

Figure 5.4: The `s_if` predicate

```
fire(Delta,PrvRules,if C do A,NewDelta):-
    assert(fired(if C do A)),
    s_r(Delta,A,NewDelta),
    check_prv(NewDelta,PrvRules).
```

Figure 5.5: The `fire` predicate

Figure 5.6 shows the application of force-rules mirroring Def. 5.1.12. It consists executing all the rules that can be fired, *i.e.* checking that the partially grounded events in the rules can be unified with the events in the list `Events`, checking with predicate `s_l` that the condition `C` holds in state `Delta` and that the events that fire the rules are not ignored by any ignore-rule. Then, rule

execution is performed by `s_prime_r` predicate by applying the actions of the rules fired, checking that the results should not be prevented and firing new if-rules if it is the case.

```
s_force(Delta,Events,IfRules,IgnRules,PrvRules,FrcRules,
  NewEvents,NewDelta):-
     findall([FE,A],(member(force FE on E if C do A,FrcRules),
       subset2(E,Events),s_l(Delta,C),
       \+ ignored(Delta,Events,E,IgnRules)),EAs),
     findall(Ev,member([Ev,Ac],EAs),Es), append(Es,Events,NewEvents),
     findall(Ac,member([Ev,Ac],EAs),As),s_prime_r(Delta,IfRules,
       PrvRules,As,NewDelta).
```

Figure 5.6: The `s_force` predicate

As figure 5.7 shows, the application of ECA-rules, mirroring Def. 5.1.11, consist on executing all the rules that can be fired, *i.e.* checking that the partially grounded events in the rules can be unified with the events in the list `Events`, checking with predicate `s_l` that the condition `C` holds in state `Delta` and that the events that fire the rules are not ignored by any ignore-rule. Then, rule execution is performed by `s_prime_r` predicate as before.

```
s_eca(Delta,Events,ECARules,IfRules,IgnRules,PrvRules,NewDelta):-
     findall(A,(member(on E if C do A,ECARules),subset2(E,Events),
          s_l(Delta,C),\+ ignored(Delta,Events,E,IgnRules)),As),
     s_prime_r(Delta,IfRules,PrvRules,As,NewDelta).
```

Figure 5.7: The `s_eca` predicate

5.2 Example of Concurrency: Soup Bowl Lifting

In this section we present an example of how to use the $\mathcal{I}$ language in order to specify a variation of a problem about concurrent action: the Soup Bowl Lifting problem [Gelfond et al., 1991]. We consider a situation where a soup bowl has to be lifted by two (physical) agents; one lifting from the right-hand side and the other one from the left-hand side. If both sides are not lifted simultaneously then the soup spills.

As introduced in Def. 5.1.6, the order in which the rules are declared is important since they are executed in the order they are declared. We do not obtain the same effect with rules 5.7, 5.8 and 5.9 (*spilled* does not hold after the bowl is lifted from both sides simultaneously) as with rules 5.9, 5.7 and 5.8 (*spilled* holds even if lifted from both sides simultaneously).

on *liftLeft* **if** *onTable* **do** `add`(*spilled*) (5.7)

on *liftRight* **if** *onTable* **do** `add`(*spilled*) (5.8)

on *liftLeft*, *liftRight* **if** *onTable* **do** `del`(*spilled*)`,``del`(*onTable*) (5.9)

Rules 5.7 and 5.8 specify that the soup is spilled whenever the bowl is lifted either from the right-hand side or the left-hand side alone. However, rule 5.9 avoids the spill effect whenever both events are done simultaneously. However, with rules 5.9, 5.7 and 5.8, we do not obtain the desired result since the *spilled* formula may be added after executing the rule that removes *spilled*.

To prevent the bowl from spilling, we may add the following rule to rules 5.7-5.9:

prevent *spilled* **if** *true* (5.10)

However, adding the following rules instead would also prevent the bowl from being lifted since ignoring one event will prevent the combined events from being considered.

ignore *liftLeft* **if** *true* (5.11)

ignore *liftRight* **if** *true* (5.12)

Contrarily, if we add rule 5.13 to rules 5.7-5.9, we prevent the bowl from being lifted from both sides simultaneously but not prevent it from being lifted from just one side since we are only ignoring the events if they occur together.

ignore *liftLeft*, *liftRight* **if** *true* (5.13)

This basic example illustrates the use of $\mathcal{I}$.

5.3 Applied Example: Bank

In this section we introduce an example of a banking institution in which agents are allowed to do certain operations with money. The operations in our bank are depositing, withdrawing and transferring. In our example we have two types of accounts called a and b owned by two different agents. In order to perform an operation in one of these accounts both agents have to *simultaneously* make the proper request.

Type a accounts have the limitation that no withdrawals, transfers and debits are allowed if the account haves a negative balance. However, if the account holder also has an account of type b with enough money then the necessary amount is automatically transferred to the account with negative credit and a fee is debited.

Type b accounts have the following limitations:

1. They cannot have a negative balance. All transactions that would cause a negative balance are rejected.

2. Withdrawing from or depositing to these accounts is not allowed[1].

Rule 5.14 specify the effects of opening an account of type T to agents $A1$ and $A2$ with an amount M of credit if another account of the same type with the same owners is not already opened.

$$\begin{array}{ll} \textbf{on} & \mathit{open_account}(\mathit{Id}, A1, A2, T, M) \\ \textbf{if} & \texttt{not}(\mathit{account}(\mathit{Id}, A1, A2, T, _)) \wedge \texttt{not}(\mathit{account}(\mathit{Id}, A2, A1, T, _)) \\ \textbf{do} & \texttt{add}(\mathit{account}(\mathit{Id}, A1, A2, T, M))\texttt{,} \\ & \texttt{add}(\mathit{inst}(\mathit{open_account}(\mathit{Id}, A1, A2, T, M), \mathit{Time})) \end{array} \tag{5.14}$$

Rule 5.15 specify the effect of withdrawing a given quantity M_q of money from a given account due to the simultaneous request of both owners of the account. The rules in the action section calculate the new credit for the account and modifies its value by removing the old credit and adding the new one. Likewise, a rule for the effects of depositing may also be specified.

$$\begin{array}{ll} \textbf{on} & \mathit{withdraw}(A1, \mathit{Id}, M_q), \mathit{withdraw}(A2, \mathit{Id}, M_q) \\ \textbf{if} & \mathit{account}(\mathit{Id}, A1, A2, T, M) \\ \textbf{do} & M2 = M - M_q\texttt{,}\texttt{del}(\mathit{account}(\mathit{Id}, A1, A2, T, M))\texttt{,} \\ & \texttt{add}(\mathit{account}(\mathit{Id}, A1, A2, T, M2))\texttt{,} \\ & \texttt{add}(\mathit{inst}(\mathit{withdraw}(A1, A2, \mathit{Id}, M_q), \mathit{Time})) \end{array} \tag{5.15}$$

Rule 5.16 specifies the effect of transferring from one account (of an agent and of a certain type) to another account possibly as payment of a certain good G: the source account is deducted the stated amount and it is added to the destination account.

$$\begin{array}{ll} \textbf{on} & \mathit{transfer}(A1, \mathit{Id}_s, \mathit{Id}_d, G, M), \mathit{transfer}(A2, \mathit{Id}_s, \mathit{Id}_d, G, M) \\ \textbf{if} & \mathit{account}(\mathit{Id}_s, A1, A2, T_s, M_s) \wedge \mathit{account}(\mathit{Id}_d, A3, A4, T_d, M_d) \\ \textbf{do} & M2_s = M_s - M\texttt{,}\texttt{del}(\mathit{account}(\mathit{Id}_s, A1, A2, T_s, M_s))\texttt{,} \\ & \texttt{add}(\mathit{account}(\mathit{Id}_s, A1, A2, T_s, M2_s))\texttt{,}M2_d = M_d + M\texttt{,} \\ & \texttt{del}(\mathit{account}(\mathit{Id}_d, A3, A4, T_d, M_d))\texttt{,} \\ & \texttt{add}(\mathit{account}(\mathit{Id}_d, A3, A4, T_d, M2_d))\texttt{,} \\ & \texttt{add}(\mathit{inst}(\mathit{transfer}(A1, A2, \mathit{Id}_s, \mathit{Id}_d, G, M), \mathit{Time})) \end{array} \tag{5.16}$$

To avoid concurrent actions affecting the same account, we use rule 5.17. In this case, only the first action is taken into account and the remaining concurrent actions are ignored.

$$\textbf{prevent}\ \mathit{account}(I, A_1, A_2, T, M) \wedge \mathit{account}(I, A_1, A_2, T, M_2)\ \textbf{if}\ M \neq M_2 \tag{5.17}$$

In our example, accounts of type a have the restriction that agents are not allowed to withdraw or transfer from a accounts with negative credit. This is achieved with rules like:

$$\textbf{ignore}\ \mathit{withdraw}(A, \mathit{Id}, _)\ \textbf{if}\ \mathit{account}(\mathit{Id}, A, _, a, M) \wedge M < 0 \tag{5.18}$$

$$\textbf{ignore}\ \mathit{transfer}(A, \mathit{Id}_s, _, _, _)\ \textbf{if}\ \mathit{account}(\mathit{Id}_s, A, _, a, M) \wedge M < 0 \tag{5.19}$$

[1]Notice that transferring from or to these accounts is still allowed.

Accounts of type b also have some restrictions. First, they cannot go into negative numbers. This is achieved with the following rule:

prevent $account(Id, A1, A2, b, M)$ **if** $M < 0$

Second, agents are not allowed to withdraw from accounts of type b. This is achieved by rule 5.20.

$$\textbf{ignore}\ withdraw(_, Id, _)\ \textbf{if}\ account(Id, _, _, b, _) \tag{5.20}$$

Furthermore, if an account of type a gets a negative balance then the necessary amount to avoid this situation is transferred from an account of type b. Rule 5.21 forces this type of events. Notice that a similar rule but with the order of the owners of the accounts reversed is also necessary since the owners may not appear in the same order.

$$\begin{array}{ll} \textbf{force} & transfer(A, Id_b, Id_a, a_negative, C), \\ & transfer(A2, Id_b, Id_a, a_negative, C) \\ \textbf{if} & account(Id_a, A, A2, a, C2) \wedge C2 < 0 \wedge C = -C2 \end{array} \tag{5.21}$$

5.4 Norm-Oriented Programming of Scenes

In this section, we apply $\mathcal{I}$ to the specification and enactment of scenes similarly to sections 4.2 - 4.4. We use the same formulae to program EIs of section 4.2.1. However, this time we give them semantics with $\mathcal{I}$. Normative positions define the normative state of agents without defining what their attitude towards it is, i.e. how they will actually behave. In the following section, we will use $\mathcal{I}$ to specify how the institution behaves with regards to these normative positions.

5.4.1 Providing Semantics to Normative Positions

We now provide some examples (translated from section 4.3) on how we explicitly manage normative positions of agents in $\mathcal{I}$. Recall that when specifying a normative system we need to define relationships among deontic notions.

We can confer different degrees of enforcement on MAS. We look again at those events that agents generate, *i.e.* illocutions as $ill(P, A, R, A_2, R_2, M, T)$; these may become institutional events at time T, *i.e.*: $inst(ill(P, A, R, A_2, R_2, M), T)$, supposing now is time T.

$$\begin{array}{ll} \textbf{on} & ill(P, A, R, A_2, R_2, M, T)\ \textbf{if}\ \texttt{time}(T) \\ \textbf{do} & \texttt{add}(inst(ill(P, A, R, A_2, R_2, M), T)) \end{array} \tag{5.22}$$

This rule is a translation of rule 4.4. In $\mathcal{I}$ permissions are not specified in the ECA-rules, these should be checked in ignore-rules and prevent-rules because, following $\mathcal{I}$ semantics, what is not ignored or prevented is therefore executed; we do not need to add an extra check for permissions. However, when relating attempts and prohibitions, as in rule 4.6 and the subsequent rules, we need to check both cases: the event is ignored if its not permitted or it is prohibited:

$$\textbf{ignore } I \textbf{ if } \texttt{time}(T) \texttt{ \& not}(per(I,T) : C \texttt{ \& sat}(C)) \tag{5.23}$$

$$\textbf{ignore } I \textbf{ if } \texttt{time}(T) \texttt{ \& } prh(I,T) : C \texttt{ \& sat}(C) \tag{5.24}$$

As for obligations, rule 5.25 is an example of rule that removes an obligation when it is fulfilled by an institutionalised event. Rule 5.26 shows an example of rule capturing that if certain conditions hold then an obligation to utter an (institutionalised) illocution before a given deadline D is generated.

$$\textbf{if } inst(I,T) \texttt{ \& } obl(I,T) : C \texttt{ \& sat}(C) \textbf{ do } \texttt{del}(obl(I,T) : C) \tag{5.25}$$

$$\textbf{if } conds \textbf{ do } \texttt{add}(obl(ill(P,A,R,A_2,R_2,M),T) : [T < D]) \tag{5.26}$$

Rule 5.27, translated from rule 4.9, states that if an obligation with deadline has not been fulfilled, *i.e.* there exists an obligation with a constraint associated the time, the deadline has passed, *i.e.* the current time is greater than or equal to the deadline, and we have not yet applied a sanction for that particular obligation, then we apply a sanction.

$$\textbf{if} \begin{pmatrix} obl(ill(inform,Ag_1,R,Ag_2,R',Action),T) : C \texttt{ \&} \\ (T < D) \in C \texttt{ \& time}(T2) \texttt{ \& } (T2 \geq D) \texttt{ \&} \\ \texttt{not}(sanction(obl(ill(inform,Ag_1,R,Ag_2,R',Action),T),(T < D))) \\ \texttt{ \& } credit(Ag_1,C) \texttt{ \& } credit(ei,C2) \texttt{ \& } C3 = C - 20 \texttt{ \& } C4 = C2 + 20 \end{pmatrix} \tag{5.27}$$
$$\textbf{do} \begin{pmatrix} \texttt{del}(credit(Ag_1,C)),\texttt{add}(credit(Ag_1,C3)), \\ \texttt{del}(credit(ei,C2)),\texttt{add}(credit(ei,C4)), \\ \texttt{add}(sanction(obl(ill(inform,Ag_1,R,Ag_2,R',Action),T),(T < D))) \end{pmatrix}$$

These examples show how $\mathcal{I}$ can be also used in the specification and enactment of protocols.

5.4.2 Normative Conflict Resolution

As we mentioned above, $\mathcal{I}$ has a clear semantics: what is not ignored or prevented is therefore executed. Therefore in the case an event is both expected and ignored, it will not be executed. Furthermore, in the case of rules 5.23 and 5.24, if conflicting permissions and prohibitions exist then the illocution is ignored. However, the normative positions are not removed and may lead the agents to confusion if they do not have clear that the EI will ignore events in case of inconsistency.

Although $\mathcal{I}$ has a fixed semantics, we can work around this by using the following rules: Rule 5.28 makes prohibitions prevail over permissions and obligations; Rule 5.29 makes prohibitions prevail over permissions but not over obligations; and Rule 5.30 makes prohibitions yield for permissions and obligations.

$$\textbf{ignore } I \textbf{ if } \texttt{time}(T) \texttt{ \& } prh(I,T) : C \texttt{ \& sat}(C) \tag{5.28}$$

$$\textbf{ignore } I \textbf{ if } \texttt{time}(T) \texttt{ \& } prh(I,T) : C \texttt{ \& sat}(C) \texttt{ \& not}(obl(I,T) : C' \texttt{ \& } sat(C')) \tag{5.29}$$

$$\textbf{ignore } I \textbf{ if } \texttt{time}(T) \texttt{ \& } prh(I,T) : C \texttt{ \& sat}(C) \texttt{ \& not}(per(I,T) : C' \texttt{ \& } sat(C')) \texttt{ \& not}(obl(I,T) : C' \texttt{ \& } sat(C')) \tag{5.30}$$

5.5 Conclusions

In this chapter we have proposed a language with different types of rules to specify norms (with temporal aspects and arithmetical constraints) over agents' simultaneous speech acts and to specify preventive and corrective actions that the system has to perform in each case. We have also proposed ignoring speech acts and preventing states of affairs as preventive actions and forcing speech acts and sanctioning as corrective actions. Finally, we have provided an interpreter for the language that we show in its entirety in Appendix B.

$\mathcal{I}$ is a rule language in which concurrent events may have a combined effect and may be ignored, forced, expected or sanctioned (instead of the standard notions in deontic logics). The semantics of our formalism relies on transition systems conferring it a well-studied semantics.

We have explored the proposal of this chapter by specifying two examples of concurrency: the soup bowl lifting problem and an example of a bank as an Electronic Institution. We have explained how to use *prevent* and *ignore* rules to resolve conflicts using different criteria.

The language $\mathcal{I}$ is useful to predict a future state of affairs with an initial state and a sequence of sets of events that occur and modify the intermediate states of affairs until we reach the final one. The limitations of the language are determined by the rule engine. These limitations include the inability to plan, i.e. determine the sequence of sets of events that must occur in order to reach a given state of affairs from a given initial state, or post-dicting, i.e. determine the previously unknown facts in a partial initial state given a final state and the sequence of sets of events that may have occurred. However, the goal of the language is to regulate a MAS and keep track of its evolution by prediction. Post-diction and planning would be interesting for a language that an agent could use for deciding which action to perform but this is not the aim of this thesis.

In this chapter, we have shown how activities can be regulated with the language presented and we have presented that $\mathcal{I}$ has a simple and fixed stance on normative conflicts but we can change the execution by adding extra ignore-rules specifying what normative positions prevail. However, this method is very rudimentary as normative positions are not removed and may lead to errors.

In [Esteva, 2003], obligations created by agent behaviour need to be fulfilled in different scenes or activities. For instance, when an agent wins an auction it

may be obliged to pay the good in a bank or a special facility for that purpose possibly using a pre-established protocol different from the auction protocol. Thus, in the next chapter, we generalise the notion of propagation of obligations to normative positions allowing actions of agents to generate normative positions in other activities and involving other agents. We specify this causal flow by means of a semi-graphical representation, the normative structure which conforms a normative layer on top of activities driven by normative positions, *e.g.* using languages presented in chapters 4 and 5. Furthermore, proposes a conflict resolution algorithm to be applied whenever a normative position is added to a protocol.

Chapter 6

A Normative Structure for Multiple Activities

In this chapter we address our research questions about how to specify norms and make them operational to regulate multiple concurrent and distributed activities. For this purpose, we propose normative structures, a computational model for the propagation of formulae among activities and we reduce the normative structure to coloured Petri nets (CPNs) to show that the verification of normative conflicts is computationally intractable. Finally, we provide an interpreter for normative transitions.

In this chapter, we deal with multiple distributed activities. However, activities are not always isolated. In [Esteva, 2003], obligations created by agent behaviour may need to be fulfilled in different scenes or activities. For instance, a buyer agent that wins an auction has the obligation to pay for the auctioned item in the near future in the premises provided for that purpose.

To deal with the distribution of activities we conceive two approaches:

Tightly-coupled distribution When an obligation is propagated from one activity to the next ones, the first activity directly sends the obligation to the others. In this approach activities are very dependent on each other as every activity should know all the possible recipients of their obligations.

Loosely-coupled distribution When an obligation is propagated from one activity to the next ones, the first activity sends the obligation to a propagator in charge of spreading the obligation among the appropriate activities. In this approach activities are less co-dependent on each other as they only need to know the pertinent propagators which, in theory, are less in number and more persistent than activities.

In this chapter, we generalise the generation of obligations in other scenes or activities to normative positions, allowing actions of agents to generate normative positions in other scenes or activities and possibly involving other agents. For instance, a buyer who ran out of credit may be forbidden from making further

offers in all the auctions it may participate until its credit has been re-established. We specify this causal flow by means of a semi-graphical representation, the normative structure (NS) which confers a normative layer on top of scenes driven by normative positions, *e.g.* using the language presented in chapter 5. The normative structure corresponds to a loosely-coupled distribution approach.

Within a NS conflicts may arise due to the dynamic nature of the MAS and the concurrency of agent's actions. For instance, an agent may be obliged and prohibited to do the same action of a protocol. Methods for conflict resolution in contradicting laws have been studied in the field of artificial intelligence and law [Sartor, 1992] and agent reasoning over conflicting normative positions has been addressed in [Kollingbaum, 2005]. However, it is not until the work in [Kollingbaum et al., 2007a] that conflicts among normative positions refined with constraints can be resolved from an institutional perspective. Normative conflicts with constraints appear when a prohibition and an obligation or permission over the same agent to perform certain action with certain constraints hold and the range of accepted and rejected values expressed by the constraints overlaps. For instance, if there exists an obligation for a particular agent to pay between 200€ and 210€ but the same agent is also forbidden to pay quantities greater than 100€ – *e.g.* because it is not desirable that agents make payments greater than its credit – then the system should decide, following pre-established criteria, if the obligation should be cancelled or if the prohibition should be modified, by changing the constraints, to allow the agent to pay between 200€ and 210€.

In this chapter, we show by translating the NS into a Coloured Petri Net (CPN) [Jensen, 1997] and borrowing some well-known theoretical result from the field of CPNs, that ensuring conflict freedom at design time is computationally intractable. Thus, we propose that conflict detection and resolution at runtime can complement formal techniques of verification used at design time. We also show in this chapter the approach of [Kollingbaum et al., 2007a] applied to our formulation of normative positions and normative structure.

6.1 Scenario

We use a supply-chain scenario in which companies and individuals come together at a virtual (electronic) marketplace to conduct business. The overall transaction procedure may be organised as six activities, represented as nodes in the diagram of Figure 6.1. They involve different participants whose behaviour is coordinated through protocols. In this scenario agents can play one of four roles: accountant (represented as *acc*), *client*, supplier (represented as *supp*) and warehouse manager (represented as *wm*). The arrows connecting the activities indicate the order activities can be enacted.

After registering at the marketplace, clients and suppliers get together in the *negotiation* activity where they agree on the terms of their transaction, *i.e.* prices, amounts of goods to be delivered, deadlines and other details. In the *contract* activity, the order becomes established and an invoice is prepared. The

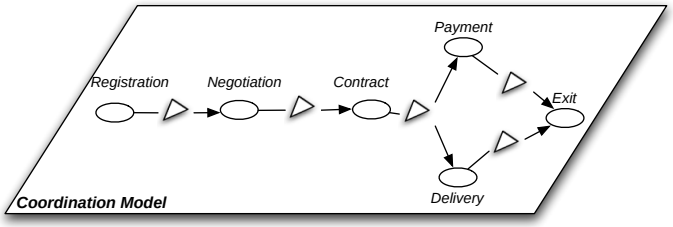

Figure 6.1: Activities of Virtual Marketplace

client will then participate in a *payment* activity, verifying his credit-worthiness and instructing his bank to transfer the correct amount of money. The supplier in the meantime will arrange for the goods to be delivered (*e.g.* via a warehouse manager) in the *delivery* activity. Finally, agents can leave the marketplace conforming to a predetermined *exit* protocol. The marketplace accountant participates in most of the activities as a trusted provider of auditing tools. In the rest of the chapter we shall build on this scenario to exemplify the notion of normative structure and to illustrate our approach to conflict detection and resolution in a distributed setting.

6.2 Normative States, Transitions and Structures

Using our basic concepts, we now provide formal definitions for normative states, transitions and structures. We first define normative states as follows:

Definition 6.2.1. A normative state Δ is a finite set of pairs $\langle \mathsf{I}, t\rangle$ (illocutions) or $\langle \mathsf{N}, t\rangle$ (normative positions), $t \in \mathbb{N}$, representing, respectively, that I was uttered at instant t and normative position N has been established at instant t (or has been active since instant t).

Normative states record both the events (what was uttered and when) and the normative positions occurring in an activity. A sample normative state of the *delivery* activity of our scenario (referred to as *dlvry*) is:

$$dlvry = \left\{ \begin{array}{c} \langle inform(sean, seller, kev, wm, deliver(wire, 200), 20), 20\rangle \\ \langle obl(inform(kev, wm, jules, client, deliver(wire, 200), 30)), 20\rangle \\ \langle prh(inform(kev, wm, jules, client, deliver(iron, 25), 30)), 20\rangle \end{array} \right\}$$

In the normative state above, agent *sean* taking up the *seller* role has delivered 200kg of wire to agent *kev* taking up the warehouse manager role wm[1]. Agent *kev* has an obligation to deliver 50kg of copper to *sean*; *kev* is, however, prohibited to deliver 25kg of iron to *jules*, another *client*. In a pair

[1]Following the terminology from [Searle, 1995] and the computational model of an activity from [García-Camino et al., 2007b], we design our MAS in order to incorporate that informing about a delivery "counts as" a factual delivery provided the items have been registered in the MAS.

$\langle p(ag, r, ag', r', \tau, t), t' \rangle \in \Delta$, it is always the case that $t = t'$, that is, the time the illocution was uttered and their time as recorded in Δ are always the same. However, in a pair $\langle obl(p(ag, r, ag', r', \tau, t)), t' \rangle \in \Delta$, t may be different from t', as t reflects the time when the illocution ought to be uttered and t' represents when the normative position was generated (the same applies to the other normative positions *per* and *prh*). Normative states evolve over time; activities are connected to one another via normative transitions that specify how utterances and normative positions in one activity (represented in its normative state) affect other activities.

Activities are not independent: illocutions uttered in some of them may have an effect in other activities. Normative transitions define the conditions under which a normative position is updated. These conditions are either utterances and/or norms associated with a given activity, which yield a *normative command*, *i.e.* the addition or removal of a normative position, possibly in a distinct activity. We capture dynamic aspects of norms (and their relationships across distinct activities) via normative transitions, defined as follows:

Definition 6.2.2. A normative transition R is of the form:

$$
\begin{array}{lcl}
\mathsf{R} & ::= & \mathsf{V} \Rightarrow \mathsf{C} \\
\mathsf{V} & ::= & \mathsf{V}\ \texttt{\&}\ \mathsf{V} \\
 & | & \mathsf{V}\ \texttt{||}\ \mathsf{V} \\
 & | & \texttt{not}(\mathsf{V}) \\
 & | & id_s : \mathsf{D} \\
 & | & \alpha \\
\mathsf{D} & ::= & \alpha \mid \alpha : \Gamma \\
\mathsf{C} & ::= & \mathsf{C},\mathsf{C} \\
 & | & \texttt{add}(id_s : \mathsf{D}) \\
 & | & \texttt{del}(id_s : \mathsf{D})
\end{array}
$$

where α is an atomic formula possibly followed by a set of constraints (Γ), id_s is an identifier for activity s and C is a normative command.

In α (but not in $id_s : \alpha$), we capture arithmetical operations and all the special commands we have been using in the previous sections such as *time*/1 and `sat`/1.

We endow our language of Definition 6.2.2 with the usual semantics of rule-based languages [Kramer and Mylopoulos, 1992] which we informally describe below. Rules map an existing normative structure to a new normative structure where only the normative scenes change. Figure 6.2 presents a possible interpreter for normative transitions.

We extend the notion of MAS with an extra layer consisting of normative states and transitions. This layer is represented as a bi-partite graph which we call *normative structure*. A normative structure relates normative states and normative transitions specifying which normative positions are to be added to or removed from which normative states.

1. $\mathbf{s}_l(\Delta, (\mathsf{V}\ \ \&\ \ \mathsf{V}'), \sigma) \leftarrow \mathbf{s}_l(\Delta, \mathsf{V}, \sigma'), \mathbf{s}_l(\Delta, \mathsf{V}' \cdot \sigma', \sigma''), \sigma = \sigma' \cup \sigma''$
2. $\mathbf{s}_l(\Delta, (\mathsf{V}\ \ ||\ \ \mathsf{V}', \sigma) \leftarrow \mathbf{s}_l(\Delta, \mathsf{V}, \sigma)$
3. $\mathbf{s}_l(\Delta, (\mathsf{V}\ \ ||\ \ \mathsf{V}'), \sigma) \leftarrow \mathbf{s}_l(\Delta, \mathsf{V}', \sigma)$
4. $\mathbf{s}_l(\Delta, \mathtt{not}(\mathsf{V}), \sigma) \leftarrow \neg \mathbf{s}_l(\Delta, \mathsf{V}, \sigma)$
5. $\mathbf{s}_l(\Delta, \mathtt{sat}(\Gamma), \sigma) \leftarrow satisfiable(\Gamma \cdot \sigma)$
6. $\mathbf{s}_l(\Delta, id_s : \mathsf{D}, \sigma) \leftarrow \mathtt{member}((id_s : \mathsf{D}) \cdot \sigma, \Delta)$
7. $\mathbf{s}_l(\Delta, \alpha, \sigma) \leftarrow \mathtt{call}(\alpha \cdot \sigma)$
8. $\mathbf{s}_r(\Delta, (\mathsf{C}, \mathsf{C}'), \Delta'') \leftarrow \mathbf{s}_r(\Delta, \mathsf{C}, \Delta'), \mathbf{s}_r(\Delta', \mathsf{C}', \Delta'')$
9. $\mathbf{s}_r(\Delta, \mathtt{add}(id_s : \alpha), \Delta') \leftarrow \Delta' = \Delta \cup \{id_s : \alpha\}$
10. $\mathbf{s}_r(\Delta, \mathtt{add}(id_s : \alpha : \Gamma), \Delta') \leftarrow satisfiable(\Gamma),\ \Delta' = \Delta \cup \{id_s : \alpha : \Gamma\}$
11. $\mathbf{s}_r(\Delta, \mathtt{add}(id_s : \alpha : \Gamma), \Delta) \leftarrow$
12. $\mathbf{s}_r(\Delta, \mathtt{del}(id_s : \mathsf{D}), \Delta') \leftarrow \mathtt{delete}(\Delta, id_s : \mathsf{D}, \Delta')$
13. $\mathbf{s}'_r(\Delta, [\,], \Delta') \leftarrow \Delta = \Delta'$
14. $\mathbf{s}'_r(\Delta, [\mathsf{C} \mid Cs], \Delta') \leftarrow \mathbf{s}_r(\Delta, \mathsf{C}, \Delta''), \mathbf{s}'_r(\Delta''', Cs, \Delta')$
15. $\mathbf{s}^*(\Delta, Rules, \Delta') \leftarrow$
    $\mathtt{findall}(\mathsf{C}, (\mathtt{member}((\mathsf{V} \Rightarrow \mathsf{C}), Rules), \mathbf{s}_l(\Delta, \mathsf{V}), Cs),$
    $\mathbf{s}'_r(\Delta, Cs, \Delta')$

Figure 6.2: An Interpreter of Normative Transitions

Definition 6.2.3. A normative structure is a labelled bi-partite graph $\langle N, E, \mathcal{L}^{in}, \mathcal{L}^{out} \rangle$ where

- $N = S \cup B$, S being a set of normative states and B a set of normative transitions
- $E = A^{in} \cup A^{out}$, $A^{in} \subseteq S \times B$ are the input arcs and $A^{out} \subseteq B \times S$ are output arcs of normative transitions
- $\mathcal{L}^{in} : A^{in} \mapsto \mathsf{D}$ is a labelling function, assigning an illocution or normative position to an input arc
- $\mathcal{L}^{out} : A^{out} \mapsto \mathsf{N}$ is a labelling function, assigning a normative position to an output arc

such that:

1. $\forall\, b \in B, \exists\ a \in A^{in} : [b = (\mathsf{V} \Rightarrow \mathsf{C})\ \wedge\ \mathsf{V} = (s : D)\ \wedge\ s \in S\ \wedge\ D \in s] \rightarrow a = (s, b)\ \wedge\ \mathcal{L}^{in}(a) = \mathsf{D}.$
2. $\forall\, b \in B, \exists\ a \in A^{out} : [b = (\mathsf{V} \Rightarrow \mathsf{C})\ \wedge\ \mathsf{C} = \bigcirc(s : N)\ \wedge\ \bigcirc \in \{\mathtt{add}, \mathtt{del}\}\ \wedge\ s \in S,\ N \in s] \rightarrow a = (b, s)\ \wedge\ \mathcal{L}^{out}(a) = \mathsf{N}.$
3. $\forall\ a \in A^{in}, \exists\ \sigma \in \Sigma : [a = (s, b)\ \wedge\ b = (\mathsf{V} \Rightarrow \mathsf{C})\ \wedge\ \mathcal{L}^{in}(a) = \mathsf{D}] \rightarrow (s : \mathsf{D}) \cdot \sigma \in \mathsf{V}.$
4. $\forall\ a \in A^{out}, \exists\ \sigma \in \Sigma : [a = (b, s)\ \wedge\ b = (\mathsf{V} \Rightarrow \mathsf{C})\ \wedge\ \mathcal{L}^{out}(a) = \mathsf{N}] \rightarrow (s : \mathsf{N}) \cdot \sigma \in \mathsf{C}.$

The first two requirements ensure that every utterance and every normative position on the left-hand side of a normative transition labels an arc entering a normative transition in the normative structure, and that the normative position on the right-hand side labels the corresponding outgoing arc. Requirements three and four ensure that labels from all incoming arcs are used in the left-hand side of the normative transition, and that the labels from all outgoing arcs are used in the right-hand side of the normative transition that these arcs leave.

The formal semantics of normative structures is defined via a mapping to Coloured Petri Nets in Section 6.3.1. Here we give the intuitive semantics of normative transition by describing how they change a normative state of a normative structure yielding a new normative structure. Each rule is triggered once for each substitution that unifies the left-hand side V of the rule with a normative state. An utterance or a normative position on the left-hand side of a rule holds iff it unifies with an utterance or normative position appearing in the normative state. Every time a rule is triggered, the update specified on the right-hand side of that rule is carried out, thus adding or removing a normative position from a normative state. Conflicts may arise after the addition of normative positions: if a conflict arises, we make use of an algorithm to decide whether to ignore the new normative position or to "adjust" old normative positions to avoid conflicts. We explain this in more detail in Section 6.4.

6.2.1 Example

We now present four normative transitions from our scenario of Section 6.1. The first rule illustrates how one single normative state can be modified. The second rule makes reference to more than one normative state in its left-hand side. Finally, rules 3 and 4 illustrate a "normative flow" whereby the appearance of a new obligation in one normative state generates other obligations in different normative states.

In our scenario, during the *negotiation* activity (represented as $ngtn$), a request of a buyer B to a seller S to sell an amount A of item It at price P leads to the introduction of a normative position in the same normative state. The normative position is a permission on the seller S to accept the request from B. This is formalised by the following rule:

$$(ngtn : request(B, buyer, S, seller, sell(It, A, P), T_1)$$
$$\Rightarrow$$
$$\texttt{add}(ngtn : per(accept(S, seller, B, buyer, sell(It, A, P), T_2)))$$

In the *negotiation* activity, if a seller S accepts the offer of a buyer B to sell an amount A of item It at price P and in the *payment* activity (represented as $pmnt$) the same buyer has paid P to seller S, then that introduces into the *delivery* activity an obligation on the seller agent to deliver the sold item It.

This is captured by the following normative transition:

$$\left(\begin{array}{l}(ngtn : accept(S, seller, B, buyer, sell(It, A, P), T_1)),\\ (pmnt : inform(B, buyer, S, seller, pay(P), T_2))\end{array}\right)$$
$$\Rightarrow$$
$$\texttt{add}(dlvry : obl(inform(S, seller, B, buyer, deliver(It, A), T_3)))$$

During the *delivery* activity, the creation of an obligation on seller s_1 to deliver copper wire leads to the propagation onto the *distribution* activity (represented as *dstr*) of an obligation on distributor d_1 to deliver that amount of copper wire to s_1. We can represent this by the following rule:

$$(dlvry : obl(inform(s_1, seller, B, buyer, deliver(copperwire, A), T_1)))$$
$$\Rightarrow$$
$$\texttt{add}(dstr : obl(inform(d_1, dstror, s_1, seller, deliver(copperwire, A), T_2)))$$

Finally, during the *distribution* activity, the creation of an obligation on the distributor d_1 to deliver some copper wire allows the propagation onto the *manufacture* activity (represented as $mnfc$) an obligation on manufacturer m_1 to deliver that amount of copper wire to d_1. This is captured by the rule below:

$$dstr : obl(inform(d_1, dstror, S, seller, deliver(copperwire, A), T_1))$$
$$\Rightarrow$$
$$\texttt{add}(mnfc : obl(inform(m_1, mnfer, d_1, dstror, deliver(copperwire, A), T_2)))$$

We show in Figure 6.3 a diagrammatic representation of how activities relate and the flow of normative positions within a normative structure: as agents utter

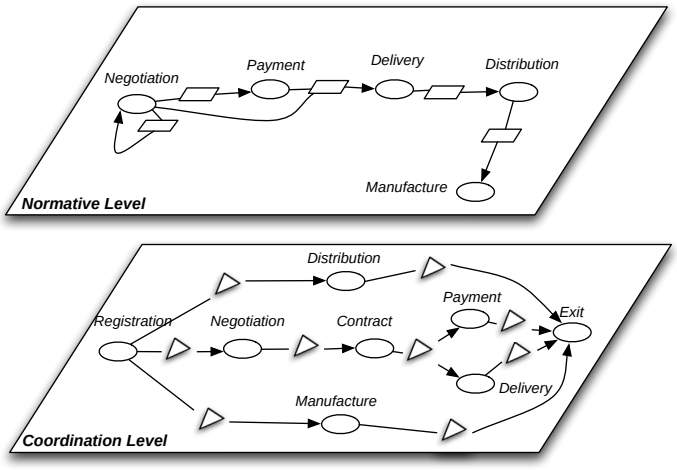

Figure 6.3: Activities and Normative Structure

illocutions during activities, normative positions arise. Utterances and normative positions are combined in normative transitions, causing the flow/propagation of normative positions between activities.

6.3 Formalising Conflict-Freedom

In this Section we introduce basic concepts of Coloured Petri Nets (CPNs). We map normative structures to CPNs and analyse their properties. We assume a basic understanding of ordinary Petri nets and refer readers to [Jensen, 1997] for more details.

CPNs combine the features of Petri nets with those of functional programming languages. Firstly, Petri nets provide the primitives for the description of the synchronisation of concurrent processes. As noticed in [Jensen, 1997], CPNs' semantics builds upon true concurrency, instead of interleaving. True concurrency semantics is more natural to work with as this is how we envisage the connection between the activity (coordination) and the normative levels of a multi-agent system. Secondly, the functional programming languages used by CPNs provide primitives for the definition of data types and the manipulation of their data values. Thus, we can readily translate expressions of a normative structure into CPNs. Thirdly, CPNs have a large number of formal analysis methods and tools through which properties of CPNs can be proved. CPNs thus provide us with all the necessary features to formally reason about normative structures given that an adequate mapping is provided.

Similarly to Petri nets, the states of a CPN are represented by means of *places*. But unlike Petri nets, each place has an associated *data type* determining the kind of data the place may contain. A state of a CPN is called a *marking*, consisting of a number of tokens positioned in individual places. Each token carries a data value which belongs to the type of the corresponding place. In general, a place may contain two or more tokens with the same data value. Thus, a marking of a CPN is a function which maps each place into a multi-set[2] of tokens of the correct type. One often refers to the token values as token *colours* and one also refers to the data types as colour sets. The types of a CPN can be arbitrarily complex.

The actions of a CPN are represented by means of *transitions*. An incoming arc into a transition from a place indicates that the transition may remove tokens from the corresponding place while an outgoing arc indicates that the transition may add tokens. The exact number of tokens and their data values are determined by the arc expressions, which are encoded using the programming language chosen for the CPN. A transition is *enabled* in a CPN iff all the variables in the expressions of its incoming arcs are bound to some value(s) (each one of these bindings is referred to as a *binding element*). If so, the transition may *occur* by removing tokens from its input places and adding tokens to its output places. In addition to the arc expressions, it is possible to attach a boolean *guard* expression (with variables) to each transition. The concepts above come together in the following definition of CPN:

Definition 6.3.1. A CPN is a tuple $Net = \langle \Sigma, P, T, A, N, C, G, E, I \rangle$ where:

1. Σ is a finite set of non-empty types, also called colour sets;

[2] A *multi-set* is an extension of ordinary sets which allows *multiple occurrences* of an element.

2. P is a finite set of places;
3. T is a finite set of transitions;
4. A is a finite set of arcs;
5. N is a node function defined from A onto $P \times T \cup T \times P$;
6. C is a colour function from P onto Σ;
7. G is a guard function from T onto expressions;
8. E is an arc expression function from A onto expressions;
9. I is an initialisation function from P onto closed expressions;

The informal explanations for the enabling and occurrence rules given above should help us to understand the behaviour of a CPN. Similar to ordinary Petri nets, the concurrent behaviour of a CPN is based on the notion of *step*. Formally, a step is a non-empty and finite multi-set over the set of all binding elements. Let step $\mathcal{S}$ be enabled in a marking $\mathcal{M}$. Then, $\mathcal{S}$ may *occur*, changing the marking $\mathcal{M}$ to $\mathcal{M}'$. Moreover, we say that marking $\mathcal{M}'$ is *directly reachable* from marking $\mathcal{M}$ by the occurrence of step $\mathcal{S}$, and we denote it by $\mathcal{M}[\mathcal{S} > \mathcal{M}'$.

A *finite occurrence sequence* is a finite sequence of steps and markings: $\mathcal{M}_1[\mathcal{S}_1 > \mathcal{M}_2 \ldots \mathcal{M}_n[\mathcal{S}_n > \mathcal{M}_{n+1}$ such that $n \in \mathbb{N}$ and $\mathcal{M}_i[\mathcal{S}_i > \mathcal{M}_{i+1}$ $\forall\, i \in \{1, \ldots, n\}$. The set of all possible markings reachable for a net *Net* from a marking $\mathcal{M}$ is called its *reachability set*, and is denoted as $R(Net, \mathcal{M})$.

6.3.1 Mapping Normative Structures to Coloured Petri Nets

We propose a mapping "$\mapsto$" from a simpler version of NSs without constraints, disjunction nor negation to CPNs, to provide semantics for the NSs and to prove properties about NSs using well-known results from CPNs. Our mapping makes use of correspondences between normative states and CPN places, normative transitions and CPN transitions and finally between arc labels and CPN arc expressions. Namely:

$$\begin{array}{rcl} S & \mapsto & P \\ B & \mapsto & T \\ \mathcal{L}^{in} \cup \mathcal{L}^{out} & \mapsto & E \end{array}$$

The set of types is the singleton set containing the colour NP (*i.e.* $\Sigma = \{\mathsf{NP}\}$). We use CPN-ML syntax [Christensen and Haagh, 1996] to structure this complex type as follows:

```
color NPT = with Obl | Per | Prh | NoMod
color IP  = with inform | declare | offer
color UTT = record
              illp   : IP
```

```
              ag1,
              role1,
              ag2,
              role2  : string
              content: string
              time   : int
color NP  = record
              mode   : NPT
              illoc  : UTT
```

Modelling illocutions as normative positions without modality (`NoMod`) is a "trick" to ensure that sub-nets can be combined as explained below. Arcs are mapped almost directly. A is a finite set of arcs and N is a node function, such that $\forall\, a \in A\ \exists\, a' \in A^{in} \cup A^{out}.\, N(a) = a'$. The initialisation function I is defined as $I(p) = \Delta_s$ ($\forall\, s \in S$ where p is obtained from s using the mapping, where $s = \langle id, \Delta_s \rangle$). Finally, the colour function C assigns the colour `NP` to every place: $C(p) = \texttt{NP}$ ($\forall\, p \in P$). We do not make use of the guard function G.

6.3.2 Properties of Normative Structures

Having defined the mapping from normative structures to Coloured Petri Nets, we now look at properties of the latter which help us understand the complexity of conflict detection. One question we would like to answer is, whether at a given point in time, a given normative structure is conflict-free. Such a snapshot of a normative structure corresponds to a marking in the mapped CPN:

Definition 6.3.2. A marking $\mathcal{M}_i$ is conflict-free if $\neg\, \exists\, p \in P.\ \exists\, np_1, np_2 \in \mathcal{M}_i(p)$ such that $np_1.mode = \texttt{Obl}$ and $np_2.mode = \texttt{Prh}$ and $np_1.illoc$ and $np_2.illoc$ unify.

Another interesting question is whether a conflict will occur from such a snapshot of the system by propagating the normative positions. In order to answer this question, we first translate the snapshot of the normative structure to the corresponding CPN and then *execute* the finite *occurence sequence* of markings and steps, verifying the conflict-freedom of each marking as we go along:

Definition 6.3.3. Given a marking $\mathcal{M}_i$, a finite occurrence sequence $\mathcal{S}_i, \mathcal{S}_{i+1}, ..., \mathcal{S}_n$ is called conflict-free, if and only if $\mathcal{M}_i[\mathcal{S}_i > \mathcal{M}_{i+1} \ldots \mathcal{M}_n[\mathcal{S}_n > \mathcal{M}_{n+1}$ and $\mathcal{M}_k$ is conflict-free for all k such that $i \leq k \leq n+1$.

However, the main question we would like to investigate is whether a given normative structure is *conflict-resistant*, that is, whether or not the agents enacting the MAS are able to bring about conflicts through their actions. When we factor in arbitrary actions (or utterances) from autonomous agents, we lose determinism.

Having mapped the normative structure to a CPN, we now add CPN models of the agents' interactions. Each form of agent interaction (*i.e.* each activity) can

be modelled using CPNs along the lines of the work described in [Cost et al., 2000]. These non-deterministic CPNs "feed" tokens into the CPN that models the normative structure, thereby introducing non-determinism into the combined CPN. Figure 6.4 shows the evolution of a CPN. The lower half of figures 6.4(a)-6.4(c) shows part of a CPN model of an agent protocol where the arc denoted with '1' represents an illocution by an agent. The target transition of this arc not only

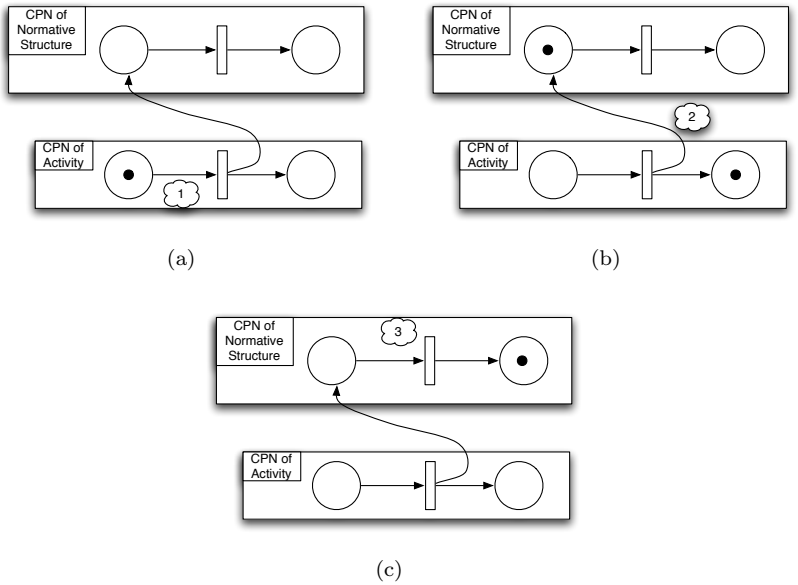

Figure 6.4: Evolution of CPNs for an Activity and a Normative Structure

moves a token on to the next state of this CPN, but also places a token in the place corresponding to the appropriate normative state in the CPN model of the normative structure (via arc '2'). Transition '3', for instance, may propagate the token in form of an obligation. Thus, from a given marking, many different sequences are possible depending on the agents' actions. We make use of a reachability set R to define a situation in which agents cannot cause conflicts.

Definition 6.3.4. A marking $\mathcal{M}_i$ is conflict-resistant iff all markings in $R(N, \mathcal{M}_i)$ are conflict-free.

Checking conflict-freedom of a marking can be done in polynomial time by checking all places of the CPN for conflicting tokens. Conflict-freedom of an occurrence sequence in the CPN that represents the normative structure can also be done in polynomial time since such a sequence is deterministic given a snapshot.

Whether or not a normative structure is designed safely corresponds to checking the conflict-resistance of the initial marking $\mathcal{M}_0$. Verifying conflict-resistence

of a marking becomes a very difficult task. It corresponds to the reachability problem in a CPN: "can a state be reached or a marking achieved, that contains a conflict?". This reachability problem is known to be NP-complete for ordinary Petri nets [Murata, 1989] and since CPNs are functionally identical, one cannot expect to verify conflict-resistence of a normative structure at design time within a reasonable amount of time. Therefore, one can only verify the conflict-freedom of the initial *state* of the normative structure and distributed, run-time mechanisms are needed to ensure that a normative structure maintains consistency.

6.4 Resolving Normative Conflicts in Run-time

In this section, norms refer to illocutions (as opposed to arbitrary actions) – normative conflicts arise when an illocution is simultaneously obliged and prohibited. The terms deontic *conflict* and deontic *inconsistency* have been used interchangeably in the literature to refer to situations in which actions are simultaneously associated with different modalities. However, in this thesis we adopt the view of [Elhag et al., 2000] in which the authors suggest that a deontic inconsistency arises when an action is simultaneously permitted and prohibited – since a permission may not be acted upon, no conflict actually occurs. The situation when an action is simultaneously obliged and prohibited is, however, a normative conflict, as both obligations and prohibitions influence agents' behaviours in a conflicting fashion.

6.4.1 Conflict Detection

We use *unification* of first-order terms [Apt, 1997, Fitting, 1990] as a means to detect and resolve conflicts between normative positions. Unification is based on the concept of substitution (*viz.* Def. 4.1.6 and 4.1.8), that is, the set of values for variables in a computation.

Definition 6.4.1. A conflict arises between $\alpha : \Gamma$ and $\alpha' : \Gamma'$ under a substitution σ, denoted as **conflict**$(\alpha : \Gamma, \alpha' : \Gamma', \sigma)$, iff the following conditions hold:

1. Either
   - 1.1. $\alpha = per(\bar{\mathsf{I}}), \alpha' = prh(\bar{\mathsf{I}}')$ or;
   - 1.2. $\alpha = obl(\bar{\mathsf{I}}), \alpha' = prh(\bar{\mathsf{I}}')$
2. $\bar{\mathsf{I}} \cdot \sigma = \bar{\mathsf{I}}' \cdot \sigma, satisfy((\Gamma \cup \Gamma') \cdot \sigma)$

Two conditions are tested: the first one checks that the various components of a possibly conflicting normative position unify; the second one checks that the constraints associated with the normative positions are satisfiable.

6.4.2 Conflict Resolution

Our work presented in [Kollingbaum et al., 2007a] is the basis of the work introduced in this section. We now formally define how the curtailment of normative positions takes place. It is important to notice that the curtailment of a normative position creates a new (possibly empty) set of curtailed normative positions.

In order to curtail $\alpha : \Gamma$ thus avoiding any overlapping of values its variables may have with those variables of $\alpha' : \Gamma'$, we must "merge" the negated constraints of $\alpha' : \Gamma'$ with those of $\alpha : \Gamma$. Additionally, in order to ensure the appropriate correspondence of variables between $\alpha : \Gamma$ and $\alpha' : \Gamma'$ is captured, we must apply the substitution σ obtained via **conflict**$(\alpha : \Gamma, \alpha' : \Gamma', \sigma)$ on the merged negated constraints.

Definition 6.4.2. Relationship **curtail**$(\alpha : \Gamma, \alpha' : \Gamma', \Delta)$ holds iff Δ is a possibly empty and finite set of normative positions obtained by curtailing $\alpha : \Gamma$ with respect to $\alpha' : \Gamma'$. The following cases arise:

1. If **conflict**$(\alpha : \Gamma, \alpha' : \Gamma', \sigma)$ does not hold then $\Delta = \{\alpha : \Gamma\}$, that is, the set of curtailments of a non-conflicting norm $\alpha : \Gamma$ is $\alpha : \Gamma$ itself.

2. If **conflict**$(\alpha : \Gamma, \alpha' : \Gamma', \sigma)$ where $\Gamma = \{\gamma_0, \ldots, \gamma_n\}$ and $\Gamma' = \{\gamma'_0, \ldots, \gamma'_m\}$ holds, then $\Delta = \{\alpha : \Gamma^c_0, \ldots, \alpha : \Gamma^c_m\}$, where $\Gamma^c_j = \Gamma \cup \neg(\gamma'_j)$, $0 \leq j \leq m$.

By combining the constraints of $\nu = \alpha : \Gamma$ and $\nu' = \alpha' : \Gamma'$ where $\Gamma = \{\gamma_0, \ldots, \gamma_n\}$ and $\Gamma' = \{\gamma'_0, \ldots, \gamma'_m\}$, we obtain the curtailed norm $\nu^c = \alpha : \Gamma \cup \neg(\Gamma')$. The following equivalences hold:

$$\alpha \wedge \textstyle\bigwedge_{i=0}^{n} \gamma_i \wedge \neg(\bigwedge_{j=0}^{m} \gamma'_j \cdot \sigma) \equiv \alpha \wedge \bigwedge_{i=0}^{n} \gamma_i \wedge (\bigvee_{j=0}^{m} \neg\gamma'_j \cdot \sigma)$$

That is, $\bigvee_{j=0}^{m}(\alpha \wedge \bigwedge_{i=0}^{n} \gamma_i \wedge \neg(\gamma'_j \cdot \sigma))$. This shows that each constraint of ν' leads to a possible solution for the resolution of a conflict and a possible curtailment of ν. The curtailment thus produces a set of curtailed norms $\nu^c_j = \alpha \wedge \bigwedge_{i=0}^{n} \gamma_i \wedge \neg\gamma'_j \cdot \sigma, 0 \leq j \leq m$. Although each of the ν^c_j, $0 \leq j \leq m$, represents a solution to the norm conflict, we advocate that *all* of them have to be added to Δ in order to replace the curtailed norm. This would allow a preservation of as much of the original scope of the curtailed norm as possible.

During the formation of a conflict-free Δ', the institution has to choose which normative position to curtail in case of a conflict. In order to express such a choice, we introduce the *curtail-rules* that determine, given a pair of normative positions, which normative position to curtail.

Definition 6.4.3. A **curtail-rule** is an expression of the form:

curtail $(\alpha : \Gamma)$ **over** $(\alpha' : \Gamma')$ **if** *conditions*

The above expression means that constrained normative position $(\alpha : \Gamma)$ has to be curtailed following Definition 6.4.2 whenever conflicts with constrained normative position $(\alpha' : \Gamma')$ and *conditions* hold.

In order to add normative positions, we will use the norm adoption algorithm shown in Figure 6.5 adapted from [Kollingbaum et al., 2007a]. The first norm is simply added (line 2), otherwise the norm is checked if conflicts with any of the active norms (lines 4-5). If it is the case, it checks if there exists a curtail rule in Π that determines that the new norm has to be curtailed (lines 6-9). Otherwise it checks if there exists a curtail rule in Π that determines that the norm in state of affairs Δ has to be curtailed (lines 14-17). The function $s_l(\Delta, conds, \sigma_4))$ checks if the conditions of the curtail rule are satisfied in the state of affairs Δ returning a substitution σ_4. Its implementation is similar to the ones in Definitions 4.1.9 and 5.1.1. When changing a norm in Δ, it should be removed and the new curtailed norms added (line 20).

algorithm $adoptNorm(\alpha : \Gamma, \Delta, \Pi, ncs)$
input $\alpha : \Gamma, \Delta, \Pi$
output ncs
begin
01 $ncs := \emptyset$
02 **if** $\Delta = \emptyset$ **then** $ncs := \{\texttt{add}(\alpha : \Gamma)\}$
03 **else**
04 **for each** $(\alpha' : \Gamma') \in \Delta$ **do**
// test for conflict
05 **if** $\alpha \cdot \sigma_1 = \alpha' \cdot \sigma_1$ **and** $satisfy((\Gamma \cup \Gamma') \cdot \sigma_1)$ **then**
// test curtail rule
06 **if** (curtail $(\alpha_\pi : \Gamma_\pi)$ over $(\alpha'_\pi : \Gamma'_\pi)$ if $conds) \in \Pi$ **and**
07 $\alpha \cdot \sigma_2 = \alpha_\pi \cdot \sigma_2$ **and** $satisfy((\Gamma \cup \Gamma_\pi) \cdot \sigma_2)$ **and**
08 $\alpha' \cdot \sigma_3 = \alpha'_\pi \cdot \sigma_3$ **and** $satisfy((\Gamma' \cup \Gamma'_\pi) \cdot \sigma_3)$ **and**
09 $s_l(\Delta, conds, \sigma_4))$
10 **then**
11 **curtail**$((\alpha : \Gamma), (\alpha' : \Gamma'), \Delta'')$
12 $ncs := \{\texttt{add}(\delta''_1), \cdots, \texttt{add}(\delta''_n))$
13 **else**
// test curtail rule
14 **if** (curtail $(\alpha'_\pi : \Gamma'_\pi)$ over $(\alpha_\pi : \Gamma_\pi)$ if $conds) \in \Pi$ **and**
15 $\alpha \cdot \sigma_2 = \alpha_\pi \cdot \sigma_2$ **and** $satisfy((\Gamma \cup \Gamma_\pi) \cdot \sigma_2)$ **and**
16 $\alpha' \cdot \sigma_3 = \alpha'_\pi \cdot \sigma_3$ **and** $satisfy((\Gamma' \cup \Gamma'_\pi) \cdot \sigma_3)$ **and**
17 $s_l(\Delta, conds, \sigma_4))$
18 **then**
19 **curtail**$((\alpha' : \Gamma'), (\alpha : \Gamma), \Delta'')$
20 $ncs := \{\texttt{del}(\alpha' : \Gamma'), \texttt{add}(\alpha : \Gamma), \texttt{add}(\delta''_1), \cdots, \texttt{add}(\delta''_n)\}$
21 **endif**
22 **endif**
23 **endif**
24 **endfor**
25 **endif**
end

Figure 6.5: Norm Adoption Algorithm

The list of normative commands *ncs* resulting after the possible curtailment of a normative position is then used in each activity in order to determine what actions may be institutionalised as explained in Chapters 4 and 5. However, two problems arise during the removal of norms. If we remove a curtailed normative position ($(\alpha : \Gamma)$ in Definition 6.4.2), should we add again the conflicting normative position that was removed on curtailing? If we want to remove the effects of a curtailing normative position ($(\alpha' : \Gamma')$ in Definition 6.4.2)[3], should we up-

[3]Recall that curtailing normative positions are already removed since curtailing.

date the curtailed normative positions to no longer be affected by the removed one? A solution provided in [Kollingbaum et al., 2007b] is to store a history of additions of normative positions, roll-back to the addition to be removed and recalculate possible changes in the final state of affairs following the history. As this can be very time-consuming, searching for efficient algorithms for removal of curtailed normative positions needs to be dealt with in future work.

6.5 Conclusions

In this chapter we have addressed our research questions about how to specify norms and make them operational to regulate multiple concurrent and distributed activities. For that purpose, we have proposed the normative structure, a computational model for the propagation of formulae among activities and we have reduced the normative structure to coloured Petri nets (CPNs) to show that the verification of normative conflicts is computationally intractable. Finally, we have provided an interpreter for the rule language used in the normative structure.

The notion of normative structure is useful because it allows the separation of activities and their interrelationships. It defines another part of the institution behaviour, i.e. how the system propagates normative positions among activities.

In the next chapter, we propose an architecture to amalgamate and implement the notions presented in the previous chapters: rule-based scenes, normative structure and deontic conflict resolution.

We have to decide in which point to include the conflict resolution algorithms. We will show that in the next chapter after presenting all the levels of the proposed distributed architecture.

Chapter 7

Computational Model and Distributed Architecture

In this chapter we propose a distributed architecture for the translation of agents' speech acts into system actions over the normative state, for the propagation of norms among activities and for the resolution of normative conflicts.

This thesis aims at helping in the building of a computational realisation of norm-regulated MAS. To achieve this, in previous chapters we have proposed computational languages and algorithms. However, we need to propose a MAS architecture to integrate and implement the notions presented in the previous chapters: rule-based protocols for activities, normative structure and deontic conflict resolution.

In this chapter we present AMELI$^+$, a distributed architecture based on the distributed architecture of electronic institutions presented in [Esteva et al., 2004]. We propose this architecture to address the regulation of the behaviour of autonomous agents and the management of the normative states of the MASs, including the propagation of normative positions and the resolution of normative conflicts.

7.1 Proposed Distributed Architecture

We propose an architecture to address the regulation of the behaviour of autonomous agents and the management of the normative states of the MASs, including the propagation of normative positions and the resolution of normative conflicts. We assume the existence of a set of agents that interact in order to pursue their goals – we do not have control on these agents' internal functioning, nor can we anticipate it. We require the following features of our architecture:

Regulated – The main goal of our architecture is to restrict the effects of agent behaviour in the specified conditions without hindering the autonomy of external agents.

Open – Instead of re-programming the MAS for each set of external agents, we advocate persistent, longer-lasting MASs where agents can join and leave them. However, agents' movements may be restricted in certain circumstances.

Structured – Agents are distinguishable by their role or their goals. Likewise each instance of an agent activity, *i.e.* each multi-agent protocol execution, is also distinguishable.

Heterogeneous – We leave to each agent programmer the decision of which agent architecture include in each external agent. We make no assumption concerning how agents are implemented.

Mediatory – As we do not control external agents internal functioning, in order to avoid undesired or unanticipated interactions, our architecture should work as a "filter" of events, *e.g.* messages between agents.

Distributed – To provide the means for implementing large regulated MAS, we require our architecture to be distributed in a network and therefore spreading and alleviating the workload and the message traffic.

Norm propagative – Although being distributed, agent interactions are not isolated and agent behaviour may have effects, in the form of addition or removal of normative positions, in later interactions, possibly involving different agents. We assume that agents may query the governors for the normative positions applicable to them before deciding what action perform. However, we should notify the external agents when an applicable normative position has been propagated.

Conflict Resolutive – Some conflicts may arise due to normative positions generated as result of agent's behaviour. Since ensuring a conflict-free MAS at design time is computationally intractable, we require that resolution of normative conflicts be applied by the MAS. This approach promotes consistency since there is a unique, valid normative state established by the system instead of many different state versions due to a conflict resolution at the agent's level.

To accomplish these requirements, we extend AMELI, the architecture presented in [Esteva et al., 2004]. That architecture is divided in three layers:

Autonomous agent layer It is formed by the set of external agents taking part in the MAS.

Social layer An infrastructure that mediates and facilitates agents' interactions while enforcing MAS rules.

Communication layer In charge of providing a reliable and orderly transport service.

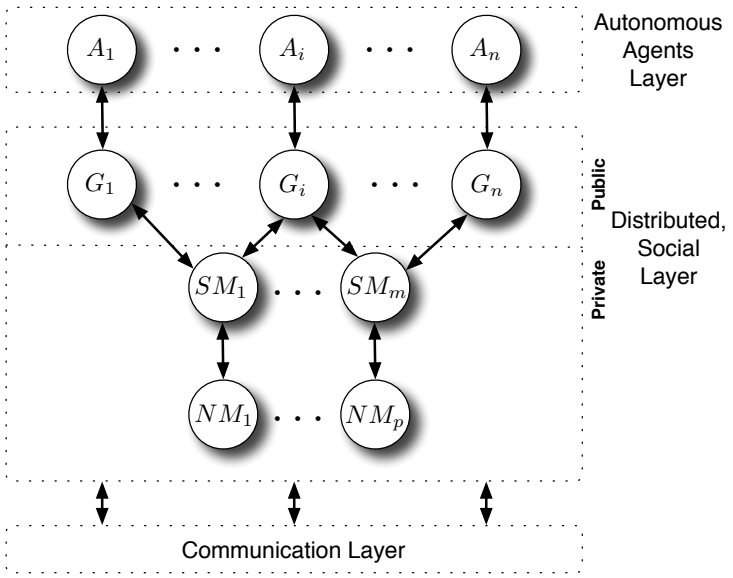

Figure 7.1: AMELI$^+$architecture

External agents intending to communicate with other external agents need to redirect their messages through the social layer which is in charge of forwarding the messages (attempts of communication) to the communication layer. In specified conditions, erroneous or illicit messages may be ignored by the social layer in order to prevent them from arriving at their addressees.

The social layer presented in [Esteva et al., 2004] is a multi-agent system itself and the agents belonging to it are called *internal agents*. We propose to extend this architecture by including a new type of agent, the normative manager (NM_1 to NM_p in fig. 7.1), and by adding protocols to accommodate this kind of agent. We call AMELI$^+$ the resulting architecture.

In AMELI$^+$, internal (administrative) agents are of one of the following types:

Governor (G) Internal agent representing an external agent, that is, maintaining and informing about its social state, deciding on whether an attempt from its external agent is valid and forwarding this attempt to the social layer. There is one per external agent.

Scene Manager (SM) Internal agent maintaining the state of the activity[1], deciding whether an attempt to communicate is valid, notifying any changes to normative managers and resolving conflicts. There is one per scene.

Normative Manager (NM) This new type of internal agent receives normative commands[2] and may fire one or more normative transition rules.

[1] Hereafter, activities are also referred to as scenes following the terminology of AMELI.

[2] Recall from section 6.2 that a normative command is the addition or removal of a normative position.

In principle, only one NM is needed if it manages all the normative transition rules. However, in order to build large MAS and avoid bottlenecks, we propose the distribution of normative transitions into several NMs.

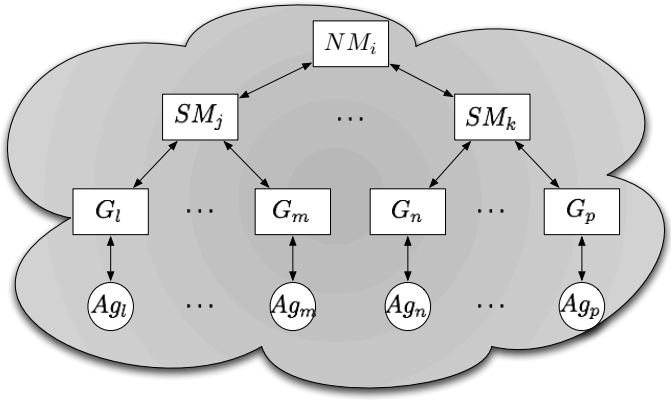

Figure 7.2: Communication channels involved in the activation of a rule

To choose the granularity of the normative layer, *i.e.*, to choose from one single NM to one NM per normative transition, is an important design decision that we leave for the MAS designers. After choosing the granularity, the NMs are assigned to handle a possibly unary set of normative transitions. We recall that each normative transition includes a rule. The SMs involved in the firing of the rules are given a reference to the NM that manages the rule, i.e. its address or identifier depending on the communication layer. External agents may join and leave activities, always following the conventions of the activities. In these cases, its governor registers (or deregisters) with the SM of that scene.

7.1.1 Social Layer Protocols

Fig. 7.2 shows the communication within the social layer – it only occurs along the following types of channels:

Agent/Governor This type of channel is used by the external agents sending messages to their respective governors to request information or to request a message to be delivered to another external agent (following the norms of the MAS). Governors use this type of channel to inform their agents about new normative positions generated.

Governor/Scene Manager Governors use this type of channel to propagate unresolved attempts to communicate or normative commands generated as a result of such attempts. SMs use this type of channel to inform governors in their scenes about new normative commands generated as a result of attempts to communicate or conflict resolution.

Scene Manager/Normative Manager This type of channel is used by SMs to propagate normative commands that NMs may need to receive and the ones resulting from conflict resolution. NMs use this channel to send normative commands generated by the application of normative transition rules.

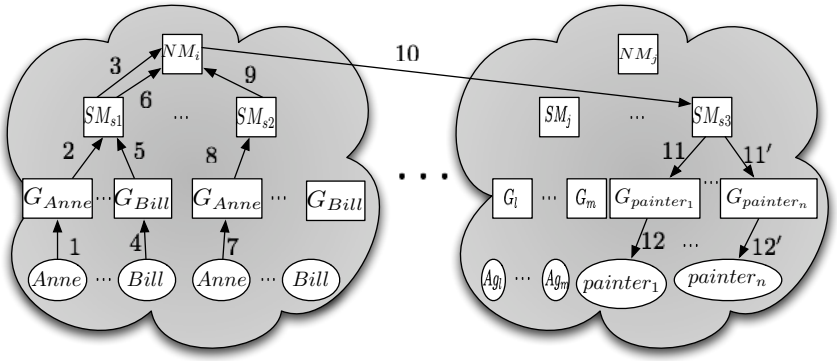

Figure 7.3: Enactment of a normative transition rule

Fig. 7.3 shows an enactment of a MAS in our architecture. Agents send attempts to governors (messages 1, 4 and 7) who, after finding out the normative commands attempts generate, propagate the new normative commands to SM_{s1} and SM_{s2} (messages 2, 5 and 8) who, in turn, propagate them to the NM (messages 3, 6 and 9). As a normative transition rule is fired in the NM, a normative command is sent to SM_{s3} (message 10). After resolving any conflicts, SM_{s3} sends the new normative commands to all the involved governors (messages 11 and 11′) who, in turn, send them to their represented agents (messages 12 and 12′).

As the figure of the previous example shows, our architecture propagates attempts to communicate (and their effects) from agents (shown on the bottom of Fig 7.3) to the NMs (shown at the top of the figure). NMs receive events from several SMs whose managed state may be arbitrarily large. Since NMs only need the normative commands that may cause any of its rules to fire, NMs subscribe only to the type of normative commands they are supposed to monitor[3]. For instance, if a rule needs to check whether there exists a prohibition to bid in a scene *auction*1 and whether there also exists the permission to bid, then the NM will subscribe to all the normative commands adding or removing prohibitions and permissions to bid in scene *auction*1.

In the following algorithms, Δ refers to essential information for the execution of the MAS, i.e. a portion of the state of affairs of the MAS that each internal agent is managing. As introduced above, depending on the type of the internal

[3]We use a publish-subscribe model to filter the amount of formulae NMs have to deal with.

agent, it manages a different portion of the state of affairs of the MAS, e.g. a governor keeps the normative state of the agent, and a scene manager keeps the state of a given scene. These algorithms define the behaviour of internal agents and are applied whenever a message msg is sent by an agent (ag_i), a governor (g_i), a SM (sm_i) or a NM (nm_i) respectively.

algorithm $G_process_att(ag_i, msg)$
input ag_i, msg
output $\emptyset$
begin
01 $new_cmmds := get_normative_commands(msg, \Delta)$
02 **foreach** $c \in new_cmmds$ **do**
03 $\quad \Delta := apply(c, \Delta)$
04 $\quad sm := scene_manager(c)$
05 $\quad send(c, ag_i)$
06 $\quad send(c, sm)$
07 **endforeach**
08 **if** $new_cmmds = \emptyset$ **then**
09 $\quad sm := scene_manager(msg)$
10 $\quad send(msg, sm)$
11 **endif**
end

Figure 7.4: Response of Governor agents to external agents' attempts

algorithm $NM_process_cmmd(sm_i, msg)$
input sm_i, msg
output $\emptyset$
begin
01 **foreach** $cmmd \in msg$ **do**
02 $\quad \Delta := apply(cmmd, \Delta)$
03 $\quad new_cmmds := get_fired_RHSs(\Delta)$
04 $\quad$ **foreach** $c \in new_cmmds$ **do**
05 $\quad\quad sm := scene_manager(c)$
06 $\quad\quad send(c, sm)$
07 $\quad$ **endforeach**
08 **foreach**
end

Figure 7.5: Response of a normative manager to a normative command

When an external agent sends to its governor an attempt to communicate (messages 1, 4 and 7 in Fig. 7.3), the governor follows the algorithm of Fig. 7.4. This algorithm checks whether the attempt to communicate generates normative commands (line 1), i.e. it is accepted[4]. This check may vary depending on the

[4]In our approach, an ignored attempt would not generate any normative command.

```
algorithm SM_process_att(g_i, msg)
input g_i, msg
output ∅
begin
01 new_cmmds := get_normative_commands(msg, Δ)
02 foreach c ∈ new_cmmds do
03    Δ := apply(c, Δ)
04    send(c, g_i)
05    foreach ⟨nm, ev⟩ ∈ subscriptions do
06       if unify(c, ev, σ) then
07       send(c, nm)
08       endif
09    endforeach
10 endforeach
11 if new_cmmds = ∅ then
12    s := scene(msg)
13    c := content(msg)
14    send(rejected(s, c), g_i)
15 endif
end
```

Figure 7.6: Response of a scene manager to a forwarded attempt

type of specification and implementation of the scenes: e.g. using Finite State Machines (FSM), as in [Esteva et al., 2004], or executing a set of rules, as in Chapters 4 and 5.

If the attempt generates normative commands (line 2), they are applied to the portion of the state of affairs the governor is currently managing creating a new partial state (line 3). These normative commands are sent to the external agent (line 5) and to the scene manager (messages 2, 5 and 8 in Fig. 7.3) in charge of the scene where the normative command should be applied (line 6). Otherwise, the attempt is forwarded to the SM of the scene the attempt was generated in (line 10).

If the governor accepts the attempt (after the check of line 1), it sends the SM a notification.The SM then applies the normative command received and forwards it to the NMs subscribed to that event (messages 3, 6 and 9 in Fig. 7.3).

However, if the governor does not take a decision, i.e. normative commands are not generated, the governor sends the attempt to the SM who should decide whether it is valid or not by following the algorithm of Fig. 7.6. This algorithm, like the one in Fig. 7.4, checks whether the received attempt generates normative commands in the current scene state, i.e. the portion of the state of affairs referring to that scene (line 1). If this is the case (line 2), they are applied to the current state of the scene (line 3) and forwarded to the governor that sent the attempt (line 4) and to the NMs subscribed to that normative commands (line 7). Otherwise (line 11), a message informing that the attempt has been rejected

```
algorithm SM_process_cmmd(nm_i, Π, msg)
input nm_i, msg
output ∅
begin
01 ncs := ∅
02 Δ' := apply(msg, Δ)
03 if inconsistent(Δ') then
04    ncs := adoptNorm(msg, Δ, Π, ncs)
05 else Δ := Δ'
06 endif
07 foreach cmmd ∈ ncs do
08    Δ := apply(cmmd, Δ)
09    foreach ⟨nm, ev⟩ ∈ subscriptions do
10       if  unify(c, ev, σ) then
11          send(c, nm)
12       endif
13    endforeach
14    foreach g ∈ governors(cmmd) do
15       send(cmmd, g)
16    endforeach
17 endforeach
end
```

Figure 7.7: Response of a scene manager to a normative command

is sent to the governor mentioned (line 14).

In both cases, if the attempt is accepted then the normative manager is notified and it follows the algorithm of Fig. 7.5 in order to decide if it is necessary to send new normative commands to other scene managers. This algorithm processes each normative command received (line 1) by applying it to the state of the NM (line 2) and checking which normative transition rules are fired and obtaining the normative commands generated (line 3). Each of them are propagated to the SM of the scene appearing in the normative command (line 6, message 10 in Fig. 7.3).

If normative commands are generated, SMs receive them from the normative manager in order to resolve possible conflicts and propagate them to the appropriate governors. In this case, the SMs execute the algorithm of Fig. 7.7. This algorithm applies the normative command received on the scene state creating a temporary state for conflict checking (line 2), then checks if the new normative command would raise an inconsistency (line 3). If this is the case, it applies the conflict resolution algorithm presented in Section 6.4, returning the set of normative commands needed to resolve the conflict (line 4). Each normative command caused by the message sent by the NM or by conflict resolution, is applied to the scene state (line 8) and it is sent to the subscribed NMs (lines 9-13) and to the governors (messages 11 and 11' in Fig. 7.3) of the agents appearing in the normative command (lines 14-16).

NMs are notified about the resolution of possible conflicts in order to check if the new normative commands fire normative transition rules. If NMs receive this notification, they follow again the algorithm of Fig. 7.5 as explained above. When governors are notified by a SM about new normative commands, they apply the normative command received to the normative state of the agent and notify to its agent about the new normative command (messages 12 and 12' in Fig. 7.3).

7.2 Conclusions

In this chapter we propose an answer for our research question about how to computationally enact distributed regulation. The answer proposed fuse our results in previous chapters obtained when answering the first four research questions. Thus, we propose a distributed architecture for the translation of agents' speech acts into system actions over the normative state, for the propagation of norms among activities and for the resolution of normative conflicts.

We base the architecture presented in this chapter in our proposal of rule-based activities presented in Chapter 5, and normative structure and conflict resolution introduced in Chapter 6.

The main contribution of this chapter is an architecture for the management of norms in a distributed manner. As a result of the partial enactment of protocols in diverse activities, normative positions generated in different activities can be used to regulate the behaviour of agents not directly involved in previous interactions.

Although scalability have improved with the distribution of activities, it may be improved by implementing peer-to-peer agent interactions. We leave for future work an study on how to regulate simultaneous actions of peer-to-peer agents.

In our approach, conflict resolution is applied at the SM level requiring all normative commands generated by a NM to pass through a SM who resolves conflicts and routes them. This feature is justified because SMs are the only agents who have a full representation of a scene and know the agents that are participating in it and which role they are enacting. For example, if a prohibition for all buyers to bid arrives at the *auction*1 activity, a SM will forward this prohibition to the governors of the agents participating in that activity with the buyer role and to the governors of all the new buyers that join that activity while the prohibition is active. An alternative approach is to apply conflict resolution at the level of governor agents, curtailing some of the normative positions of its associated external agent. However, this type of conflict resolution is more limited since a governor only maintains the normative state of an agent. For example, a case that cannot be resolved with this approach is when all agents enacting a role are simultaneously prohibited and obliged to do something, i.e. when more than one agent is involved in the conflict.

Another approach would be if governors became the only managers of normative positions; in this case they would need to be aware of all normative positions

that may affect its agent in the future, *i.e.*, they would have to maintain all the normative positions affecting any of the roles that its agent may enact in every existing scene. For instance, a governor of an agent that is not yet enacting a buyer role would also need to receive the normative positions that now applies to that role even if the agent is not in that scene or is enacting that role yet. This approach does not help with scalability since a large MAS with various scenes may generate a very large quantity of normative positions affecting agents in the future by the mere fact of their entering the MAS.

Conflict resolution is thus applied at the activity level meaning that resolution criteria involving more than one agent are now possible, as opposed to being applied at the agent level.

A central service as the one implemented in chapter 3 may become a bottleneck as the number of agents, interactions and activities grow. In opposition to the implementation of chapter 3, where the management of norms is centralised in one production system, the proposal of this chapter advocates for its distribution in scene managers (one per activity) and normative managers (managing one or more normative transitions that possibly involves more than two activities).

As we need further analysis to decide the optimal number of normative transitions to be included in normative managers, we leave this question as a design decision, *i.e.* to be answered in design time depending on the problem to be solved.

Chapter 8

Conclusions and Future Work

In this chapter we will summarise the achievements of our research and contrast them with other recent work. Finally, we will point out possible paths for advancing this work.

8.1 Conclusions

We recall the five research issues we posed in Chapter 1:

Q.1 How to specify norms that regulate a multi-agent activity.

Q.2 How to make norms operational to regulate a multi-agent activity.

Q.3 How to specify norms that handle multiple concurrent activities.

Q.4 How to make norms operational to regulate multiple concurrent and distributed activities.

Q.5 How to computationally enact distributed regulation.

We answered **Q.1** throughout Chapters 3-5 by proposing three languages and their respective interpreters. Jess norms is the first norm language dealing with time in electronic institutions. However, it does not have a formal semantics and it does not manage neither constraints nor actions. IRL is the first language to include constraint management specifying the effects of valid events. We notice that IRL has improved the aspects of constraint management and action regulation. However, IRL does not regulate simultaneous actions. Finally, $\mathcal{I}$ is the first language to include that aspect allowing, for instance, to prevent simultaneous actions from being performed.

Table 8.1 shows a comparison of the expressiveness of the languages presented in related work (introduced in Section 2.2) and those proposed in this research.

To compare normative languages we use the following features:

	Language	Features			
		Constraints	Distribution	Conc. Behaviour	Conc. Regulation
1.	CDeonticL	time	centralised	actions	monitoring
2.	Z	–	agents	actions	goals
3.	EC	time	centralised	actions	one action
4.	Rights	specification	centralised	actions	one action
5.	NoA	–	agents	actions	goals
6.	SIC	specification	centralised	actions	monitoring
7.	OCL	specification	centralised	activities	one action
8.	hyMTL	time	centralised	actions	monitoring
9.	Jess Norms	time	centralised	activities	one action
10.	IRL	management	one activity	actions	one action
11.	$\mathcal{I}$	management	one activity	actions	simultaneous actions and prevent goals

Table 8.1: Final comparison of the different norm languages

Constraints – This feature depicts the degree of constraint specification and management. We distinguish among no specification (–), specification only of time constraints (time), specification of constraints (specification) and specification and modification (management).

Distribution – This feature reflects the degree of distribution of norms. We distinguish among no distribution of norms (centralised), norms distributed in each agent (agents), and norms distributed in each activity (activities).

Concurrent Behaviour – This feature shows the degree of concurrency on actions. We distinguish among no concurrency (–), concurrent actions in one or no activity (actions), and concurrent actions in concurrent activities (activities).

Concurrent Regulation – This feature depicts the degree of regulation on concurrent actions. We distinguish among:

- no regulation of actions (–),
- no regulation of actions but regulation of goals (goals),
- just monitoring and sanctioning of actions (monitoring),
- monitoring, sanctioning and prevention of one action at a time (one action),
- monitoring, sanctioning and prevention of simultaneous actions (simultaneous actions).

Let us consider the case when a certain agent has to pay a good won in an auction before a deadline. Thus, the first feature we required for our languages is time management. Almost all approaches presented this feature, except 2 ([López y López, 2003]) and 5 ([Kollingbaum, 2005]).

We also required the second and third features for our language, namely constraint specification and management. Let us consider now that we specify restrictions on the amount of money agents may bid and under certain conditions

we can relax these constraints. For instance, agents are not allowed to make unsupported bids if they are not a regular client, *i.e.* regular clients are endowed with a certain amount of credit. To the best of our knowledge, Rights, SIC and OCL can also specify constraints but they cannot change them. Our languages, since the work of Chapter 4, not only specify constraints but also manages them allowing changes on constraints due to agents' actions or time events.

Finally, we required our language to regulate (quasi)-simultaneous actions. This feature is only present in $\mathcal{I}$, our language presented in Chapter 5. For instance, this feature allows to forbid (and to prevent if required) two robots from entering in the same cell at the same time to avoid collisions.

We answered **Q.2** by proposing two computational models that try to make norms operational in MAS. One is the centralised production system presented in Chapter 3, and the other one is presented in Section 4.1. The latter acquires the events of agents during a short period of time where events are considered simultaneous and then events are processed by the interpreter of $\mathcal{I}$, the language presented in Chapter 5.

We define normative structure (addressing **Q.3**), a model to specify and propagate that uses rules and given certain events in particular activities generates effects in other activities such as the activation or deactivation of norms.

To deal with the distribution of activities we conceive two approaches:

Tightly-coupled distribution When a normative position, *e.g.* an obligation, is propagated from one activity to the next ones, the first activity directly sends the normative position to the others. In this approach activities are very dependent on each other as every activity should know all the possible recipients of their normative positions.

Loosely-coupled distribution When a normative position is propagated from one activity to the next ones, the first activity sends the normative position to a propagator in charge of spreading the normative position among the appropriate activities. In this approach activities are less dependent as they only need to know the pertinent propagators which, in theory, are less in number and more persistent than activities.

Although rules among contexts to generate new formulae in other contexts have already been proposed [Giunchiglia and Serafini, 1994], the use of a normative structure to activate norms provides a loosely-coupling between activities allowing the latter to be specified with different languages or different semantics for obligations, permissions, and prohibitions and still be able to interoperate whenever they share the same syntax for normative positions. This partial decoupling also allows an easy distribution of activities allowing to build larger MASs. Furthermore, this decoupling is relevant to designers of heterogeneous distributed processes, and therefore electronic institution designers, that may need to interchange information and may not be aware of the other processes. This contribution also addresses partially the definition of the computational model for this kind of MAS (addressing **Q.4**).

We answered **Q.4** by proposing a computational model that tries to resolve the distributed regulation of activities with possibly conflicting norms in MAS. The distributed computational model is put together in Chapter 7 although it is partially presented in section 4.1 for regulating each activity, and in section 6.4 for propagating normative positions among activities. The resolution of normative conflicts is presented in Section 6.3.

Q.5 is answered in Chapter 7 by proposing a distributed architecture to include the results of previous questions, *i.e.* the enactment of MAS including the regulation of agent activities with norms activated as result of agent behaviour, the activation of norms among activities and the resolution of conflicts among norms in an activity at runtime. This architecture establishes the basis for building MAS enriched to enforce, propagate, and resolve conflicts in sets of norms at runtime. Furthermore, by including this architecture into EIDE, the electronic institutions framework, we will enable them to incorporate all the conceptual contributions we have proposed.

To compare models for regulated MAS we use the following features:

Openness – This feature depicts whether new heterogeneous agents may join and leave at runtime. We distinguish closed, closed and heterogeneous and open (and heterogeneous).

Regulation – This feature reflects the degree of regulation. We distinguish among: specification of protocols (protocols), specification of protocols and actions agents are expected to perform (protocols and obligations, or protocols and commitments), specification of protocols, obligations, permissions and prohibitions (protocols and norms) and specification of just norms or policies (norms).

Social Structure – We distinguish among no social structure (–), just role labels (roles), and role hierarchy.

Activity Structure – This feature depicts the degree of structure of actions. We distinguish among: no structure, just a set of actions (actions), different sets of norms apply to just a set of actions (contexts), and a separation of sets of actions (and norms) depending on its purpose (activities).

Finally, we contributed towards the declarative implementation of EIs by regulating multiple activities with rules. In opposition to the hard-coding of an interpreter of multi-agent Finite State Machine (FSM) specifications, the inclusion of a production system in each scene manager allows a more flexible and expressive specification of the protocols to be followed as we showed in Chapters 4 and 5.

The implementations of the Dutch auction introduced in Chapters 4 and 5 are a clear example that multi-agent FSMs are not expressive enough to specify temporal requirements. Thus, EIDE added a so-called constraint language that using the semantics of finite states further restricts the FSM specification. However, this language has no formal semantics and therefore it would be more

Models		Features			
		Openness	Regulation	Social Structure	Activity Structure
1.	EIs	open and heterogeneous	protocols and obligations	role hierarchy	activities
2.	MOISE$^+$	open and heterogeneous	protocols	role hierarchy	activities
3.	Commit. Insts.	open and heterogeneous	protocols and commitments	role hierarchy	activities
4.	OperA	open and heterogeneous	protocols and norms	role hierarchy	activities
5.	LGI	open and heterogeneous	norms	roles	actions
6.	EIs for VOs	open and heterogeneous	norms	roles	contexts
7.	MA Policy Arch.	open and heterogeneous	norms	roles	actions
8.	Declarative EIs	open and heterogeneous	norms	roles	activities

Table 8.2: Final comparison of the different models of regulated MAS

difficult to create reasoning about that language since we would need an ad-hoc reasoning engine.

In the AMELI version based on JADE platform [Bellifemine et al., 1999], agents participating in a scene reside in the same container kept in the same computer. However, in declarative EIs we envisage a differentiation between *spatial* and *metaphoric* agent location. Whereas the spatial location is the location of the computer that hosts an agent, the metaphoric locations are the activities (scenes) an agent is participating in. We propose that, independently from its spatial location, an agent may participate in distributed activities and metaphorically move among them just by sending messages. This proposal requires less network overhead since we do not use spatial movements among computers hosting a scene. However, the message traffic from spatially-dispersed agents towards each scene increases. We could explore the design of a hybrid system that decides to move an agent spatially or metaphorically according to certain criteria such as the network load and the number of remaining movements.

8.2 Future Work

In this section we propose future work organised according to its topic and its difficulty.

8.2.1 Improving Expressiveness

Figure 8.1 shows a comparison of the proposed languages in terms of expressiveness and distribution. On the one hand, the languages of Chapters 3-5 grow in expressiveness without dealing with distribution. On the other hand, the language of Chapter 6 deals with distribution of activities at the same degree of

expressiveness than the language presented in Chapter 4. We envisage that a natural next step would be to investigate which are the benefits of a language as expressive as $\mathcal{I}$, the one of Chapter 5, that regulates simultaneous events and as distributed as normative transitions, the one of Chapter 6, that propagates normative positions among distributed activities. We labelled this language as the language of Chapter 8.

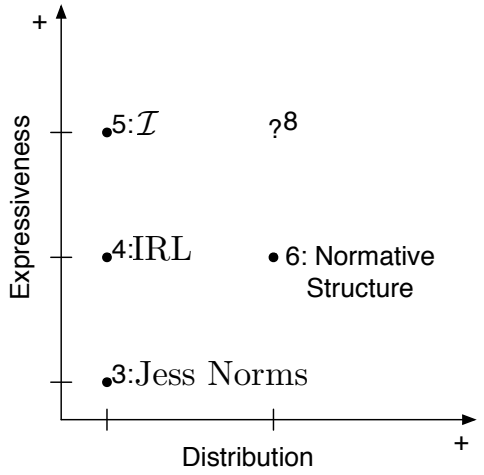

Figure 8.1: Comparison of proposed languages

One step along this direction would be to use $\mathcal{I}$ (presented in Chapter 5) as the language in the normative structure (Chapter 6). We find it interesting to explore the applications of normative structures with and without the use of this language. For instance, a normative structure using $\mathcal{I}$ could also propagate agents among activities imposing some restrictions to simultaneous agent behaviour. Therefore, normative transitions could be used to synchronise the entrance and exit of agents from activities. Furthermore, in all cases actions performed in some activities may generate facts in several activities. We think this effect as a *distributed count as* relationship [Searle, 1995] or as *distributed logical consequence*.

The state of affairs Δ' resulting after the possible curtailment of a normative position is then used in each activity in order to determine what actions may be institutionalised as explained in Chapters 4 and 5. However, two problems appear during the removal of norms. If we remove a curtailed normative position ($(\alpha : \Gamma)$ in Definition 6.4.2), should we add again the conflicting normative position that was removed upon curtailing? If we want to remove the effects of a curtailing normative position ($(\alpha' : \Gamma')$ in Definition 6.4.2)[1], should we update the curtailed normative positions to no longer be affected by the removed one? A

[1]Recall that some normative positions are already removed due to curtailing.

solution provided in [Kollingbaum et al., 2007b] is to store a history of additions of normative positions, roll-back to the addition to be removed and recalculate possible changes in the final state of affairs following the history. As this can be very time-consuming, it is a problem that need to be dealt with in future work.

In this thesis we assume that norms are static during the whole execution of the MAS and therefore, they can be provided to agents in design time. However, in a more dynamic setting, norms may vary during execution and therefore they need to be provided to agents at runtime, *e.g.* traffic lights. One interesting method to take into account is spatially distributed normative objects [Okuyama et al., 2007].

8.2.2 Electronic Institutions

We want to incorporate the new architecture to the Electronic Institutions Development Environment (EIDE). As we need further research to decide on the optimal number of normative transitions to be processed by each normative manager (*cf.* Section 7.1), we leave this question as a design decision, *i.e.* to be answered at design time depending the problem to be solved.

We are conceiving a hybrid architecture that includes both proposals of norm-management: agent norms maintained by governors and scene norms maintained by scene managers. However, a question that arises is how to automatically distinguish and separate both kind of norms into their appropriate manager.

We have contributed to the implementation of Declarative Electronic Institutions by regulating multiple activities with rules. However, to completely capture the current model of EIs, we missed some features. We would need to implement transitions of the performative structure in order to restrict the entrance and exit of agents in activities. As we mentioned above, we envisage the use of $\mathcal{I}$ in the normative structure for this purpose. As normative structures no longer would be only about norm propagation but also about propagation of agents and maybe other facts, we propose to call it *flow structure*. Likewise, we expect to include a reproduction of the concept of ontology in EIs in order to represent the typology of the objects involved in the problem domain. Having summarised this typology would ease the verification of type errors in the specification of EIs. For instance, we could decide to avoid selling short-term goods like fish in non-Dutch-like auctions by defining what is a short-term good and only allowing this kind of goods to be auctioned in Dutch auctions. While improving the work in this thesis, the performative structure was enhanced by allowing recursion, *i.e.* a node in a performative structure may now, apart from being a scene, be another performative structure. At this point, we noticed two kinds of propagation: the horizontal one that flows among different nodes and vertical propagation where a norm flows from the top level of the performative structure towards nodes in the lower level. [García-Camino et al., 2007a] was a first attempt to formalise vertical propagation of norms. However, a robust implementable model is still missing.

8.2.3 New applications using norms

In this thesis, following the EI tradition we applied in our examples the regulation of MAS to e-commerce, mainly to auctions. However, we want to find and exploit new interesting applications of norms in MAS that may lead to enrich the current model. We envisage its application in robot soccer and autonomic networking.

A question in the field of robot soccer that we find interesting is: *how robots could autonomously play soccer without a referee?* To solve this we envisage a software layer common to robots of any team. This layer would act as a middleware between the hardware layer, that deals with motors, rotors, sensors, etc., and the reasoning layer, that according with the information of the hardware layer generate an individual or team plan that is transformed in commands to the hardware layer in order to play. This middleware would follow the metaphor of institution but distributed in all robots to enforce some global rules. As a literary example, the reader may think of the hard-coding of Asimov' three laws of robotics in the "brain" of every robot. However, the programming of the reasoning layer would be available of end-users, as human coaches in the case of robot soccer. This topic requires further exploration as, for instance, norms could vary in time possibly as result of the execution of protocols. For instance, soccer rules may change from season to season due to some recurrent unwanted behaviour of players.

In the field of autonomic networking we would like to address the question: *how would the addition of obligations in policy management affect the behaviour of the global system?* For instance, some nodes in the network negotiated certain degree of quality of service (QoS) with some time constraints. Therefore the client nodes may *expect* this degree of quality from provider nodes. If the QoS contract is broken, the client may apply some punishments like choosing another provider or organising a denial-of-service attack by all the "unhappy" client nodes.

Another long-term issue is the implementation of norm reasoning in a society of agents. Some attempts have been made to improve the BDI architecture with commitments (BDI-C) [Gaertner et al., 2006]. The next step maybe the inclusion of uncertainty along the lines of [Casali et al., 2005] in order to reason about conflicts between commitments and desires. We envisage a BDI architecture capable of representing and reasoning constrained formulae along the line of our normative positions. Then, norms could be specified as beliefs therefore not needing an extra commitment component as in BDI-C.

Appendix A

UML Diagrams

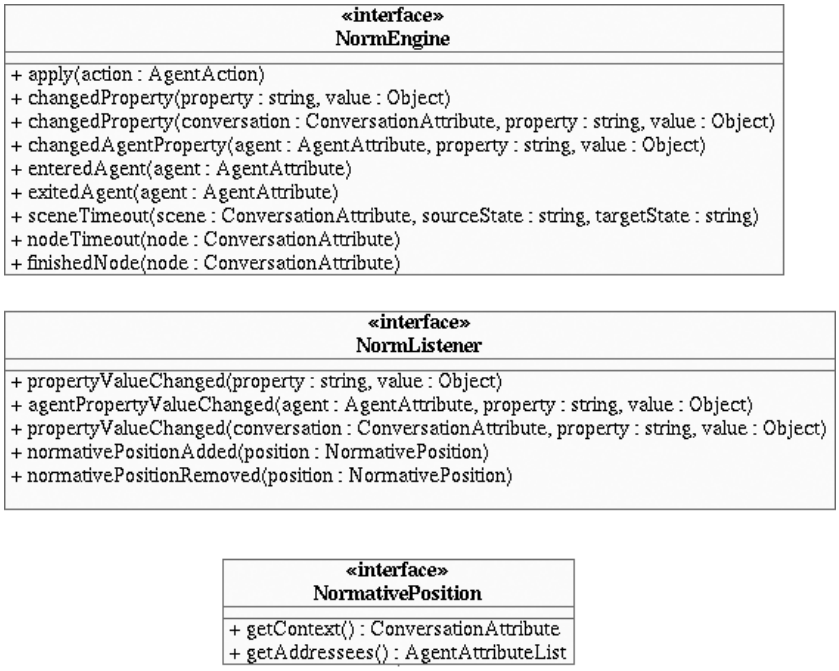

Figure A.1: Interfaces between AMELI and norm engine of Chapter 3

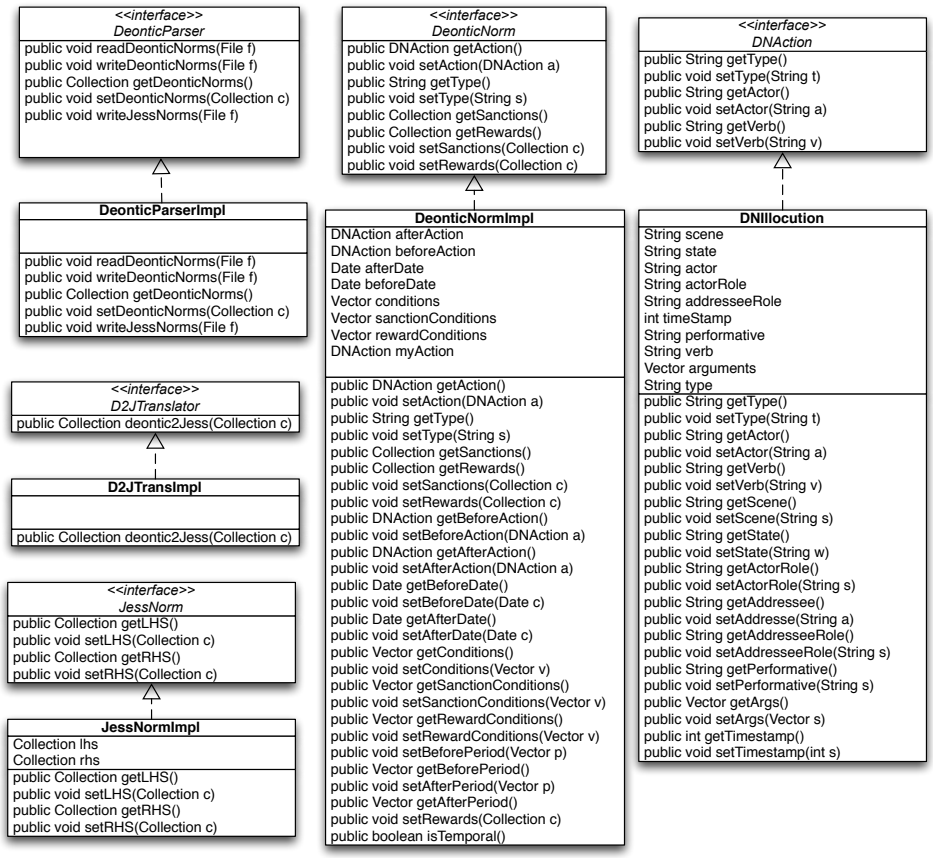

Figure A.2: UML Diagram of the Automatic Translator of Chapter 3

Appendix B

An Interpreter for $\mathcal{I}$

```
% ------------------------------------------------------------------------
% File: I-interpreter.pl
% Date: 10 March 2007
% Author: Andres Garcia-Camino -- andres@iiia.csic.es
% Description:
%   This program implements an interpreter for electronic institutions and norms
%   represented as rules.
% History:
%   version 5: Add s_if in s_star -> forward chaining is activated even there is
%                     no event
%    version 4: Fixes removal of formulae with variables
%    version 3: Adds seteq command
%    version 2: Fixes intersects(E,E') vs intersects(E',E)
%
% Instructions:
%   1. Edit the rule/2 facts to accommodate your rules
% ------------------------------------------------------------------------

:- op(170,fx,[on,prevent,ignore,force,if]).
:- op(160,xfy,[if,do,on]).
:- use_module(library(lists)).
:- dynamic fired/1.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reset
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 1. Retract all fired(_) assertions.
% 2. Assert the fired(_,_) fact so that "can_fire" does not fail if no such
% predicate exists.

reset :-
        findall(C,retract(fired(C,_)),_),
```

```
        assert(fired(false,false)).
% ------------------------------------------------------------------------

run(Delta,Events,NewDelta):-
findall(R,(rule(Id,R),R=(on _ if _ do _)),ECARules),
findall(R,(rule(Id,R),R=(ignore _ if _)),IgnRules),
findall(R,(rule(Id,R),R=(prevent _ if _)),PrvRules),
findall(R,(rule(Id,R),R=(force _ on _ if _ do _)),FrcRules),
findall(R,(rule(Id,R),R=(if _ do _)),IfRules),
s_star(Delta,Events,ECARules,IfRules,IgnRules,PrvRules,FrcRules,NewDelta).

% ------------------------------------------------------------------------

s_star(Delta,Events,ECARules,IfRules,IgnRules,PrvRules,FrcRules,NewDelta):-
reset,
s_if(Delta,IfRules,PrvRules,Delta2),
s_f(Delta2,Events,IfRules,IgnRules,PrvRules,FrcRules,NewEvents,Delta3),
s_on(Delta3,NewEvents,ECARules,IfRules,IgnRules,PrvRules,NewDelta).

% ------------------------------------------------------------------------

s_f(Delta,Events,IfRules,IgnRules,PrvRules,FrcRules,NewEvents,NewDelta):-
findall([FE,A],(member(force FE on E if C do A,FrcRules),subset2(E,Events),
s_l(Delta,C),\+ ignored(Delta,Events,E,IgnRules)),EAs),
findall(Ev,member([Ev,Ac],EAs),Es), append(Es,Events,NewEvents),
findall(Ac,member([Ev,Ac],EAs),As),s_prime_r(Delta,IfRules,PrvRules,As,NewDelta).
% ------------------------------------------------------------------------
s_on(Delta,Events,ECARules,IfRules,IgnRules,PrvRules,NewDelta):-
findall(A,(member(on E if C do A,ECARules),subset2(E,Events),
s_l(Delta,C),\+ ignored(Delta,Events,E,IgnRules)),As),
s_prime_r(Delta,IfRules,PrvRules,As,NewDelta).
% ------------------------------------------------------------------------

ignored(Delta,Events,E,IgnRules):-member(ignore E2 if C,IgnRules),
    subset2(E2,Events),
intersects(E,E2),s_l(Delta,C). % changed intersects(E2,E) by intersects(E,E2).

% ------------------------------------------------------------------------
 s_prime_r(S,_,_,[],S).
 s_prime_r(S,IfRules,PrvRules,[A|As],NewS):-
    s_r(S,A,TmpS),
    check_prv(TmpS,PrvRules),
    s_if(TmpS,IfRules,PrvRules,TmpS2),
    s_prime_r(TmpS2,IfRules,PrvRules,As,NewS).
 s_prime_r(S,IfRules,PrvRules,[_|As],NewS):-
    s_prime_r(S,IfRules,PrvRules,As,NewS).
% ------------------------------------------------------------------------
check_prv(Delta,PrvRules):-
findall(C,(member(prevent C if C2,PrvRules),s_l(Delta,C),s_l(Delta,C2)),[]).
% ------------------------------------------------------------------------
```

```
% s_l(S/+,Conditions/+):-
% - Checks if Conditions matches S
% - When checking it also assigns values to variables in Conditions
% - Conditions is a list of the terms
% ------------------------------------------------------------------------------
s_l(_,[]).                              % end of list?
s_l(S,([Atf|Atfs])):-                   % otherwise
    !,
    s_l(S,Atf),                         % check each element of the list...
    s_l(S,Atfs).
s_l(S,\+ Atf):-                         % handle negation
    !,
    \+ s_l(S,Atf).
s_l(_,seteq(L,L2)):- !, subset2(L,L2), subset2(L2,L),length(L,N),length(L2,N).
s_l(_,true):-!.    % handle true keyword
s_l(_,prolog(Goal)):-                   % handle arbitrary Prolog goal
    !,
    call(Goal).
s_l(S,Atf):-                            % atfs must be in S
    !,
    member(Atf,S).

% ------------------------------------------------------------------------------
s_r(S,[],S).                    % end of updates?
s_r(S,[Update|RHS],NewS):-      % otherwise
    !,
    s_r(S,Update,STemp),        % handle each update at a time
    s_r(STemp,RHS,NewS).
s_r(S,add(Atf),[Atf|S]):-   % add atf?
    !,
    \+ member(Atf,S).
s_r(S,add(_),S).
s_r(S,del(Atf),NewS):-      % delete atf
    !,
    member(Atf,S),
    delete(S,Atf,NewS).
s_r(S,del(_),S).
s_r(S,prolog(Goal),S):-
    !,
    call(Goal).

% ------------------------------------------------------------------------------
intersects(L,L).
intersects(L,L2):- member(A,L),member(A,L2).
intersects(A,L):-member(A,L).
% ------------------------------------------------------------------------------
subset2([],[]).
subset2([],[_|_]).
subset2([H|T],L):- member(H,L), subset2(T,L).
subset2(A,L):- member(A,L).
```

```
% ------------------------------------------------------------------------
 s_if(Delta,IfRules,PrvRules,NewDelta):-
  select_rule(Delta,IfRules,R),R\=[],
  fire(Delta,PrvRules,R,TmpDelta),
s_if(TmpDelta,IfRules,PrvRules,NewDelta).
 s_if(Delta,IfRules,_,Delta):-
  select_rule(Delta,IfRules,[]).
 s_if(Delta,IfRules,PrvRules,NewDelta):-
  s_if(Delta,IfRules,PrvRules,NewDelta).
% ------------------------------------------------------------------------
select_rule(Delta,IfRules,R):-
findall(Rule,(member(Rule,IfRules),can_fire(Delta,Rule)),Rs),
resolve(Rs,R).
% ------------------------------------------------------------------------
resolve([],[]).
resolve([H|_],H).
% ------------------------------------------------------------------------
can_fire(Delta,if C do _):-s_l(Delta,C),\+ fired(C,A).
% ------------------------------------------------------------------------
fire(Delta,PrvRules,if C do A,NewDelta):-
assert(fired(C,A)),s_r(Delta,A,NewDelta),
check_prv(NewDelta,PrvRules).
```

Bibliography

[Alberti et al., 2005] Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Sartor, G., and Torroni, P. (2005). Mapping deontic operators to abductive expectations. In *Proceedings of 1st International Symposium on Normative Multiagent Systems (NorMAS 2005), AISB 2005*, Hertfordshire, Hatfield, UK. Cited in page 19

[Alberti et al., 2003] Alberti, M., Gavanelli, M., Lamma, E., Mello, P., and Torroni, P. (2003). Specification and Verification of Agent Interactions using Integrity Social Constraints. Technical Report DEIS-LIA-006-03, Università degli Studi di Bologna. Cited in page 18

[Alchourron and Bulygin, 1981] Alchourron, C. E. and Bulygin, E. (1981). The Expressive Conception of Norms. In Hilpinen, R., editor, *New Studies in Deontic Logics*, pages 95–124, London. D. Reidel. Cited in page 11

[Apt, 1997] Apt, K. R. (1997). *From Logic Programming to Prolog.* Prentice-Hall, U.K. Cited in page 49, 50, 52, 96

[Åqvist, 1994] Åqvist, L. (1994). *Handbook of Philosophical Logic: Volume II Extensions of Classical Logic*, chapter Deontic Logic. Kluwer. Cited in page 12

[Artikis et al., 2005] Artikis, A., Kamara, L., Pitt, J., and Sergot, M. (2005). A Protocol for Resource Sharing in Norm-Governed Ad Hoc Networks. In *Declarative Agent Languages and Technologies II*, volume 3476 of *LNCS*. Springer-Verlag. Cited in page ix, 16, 17

[Bellifemine et al., 1999] Bellifemine, F., Poggi, A., and Rimassa, G. (1999). Jade - a fipa-compliant agent framework. Technical report, Telecom Italia. Part of this report has been also published in Proceedings of PAAM'99, London, April 1999, pp.97-108. Cited in page 115

[Blackburn et al., 2002] Blackburn, P., de Rijke, M., and Venema, Y. (2002). *Modal Logic*, chapter 7, pages 436–447. Cambridge University Press. Cited in page 21

[Boella and van der Torre, 2003] Boella, G. and van der Torre, L. (2003). Permission and Obligations in Hierarchical Normative Systems. In *Proceedings*

8th International Conference in AI & Law (ICAIL'03), Edinburgh. ACM. Cited in page 11

[Broersen et al., 2004] Broersen, J., Dignum, F., Dignum, V., and Meyer, J.-J. C. (2004). Designing a deontic logic of deadlines. In *7th Int. Workshop of Deontic Logic in Computer Science (DEON'04)*, pages 43–56, Portugal. Cited in page 13

[Casali et al., 2005] Casali, A., Godo, L., and Sierra, C. (2005). Graded BDI models for agent architectures. In *CLIMA V*, volume 3487 of *Lecture Notes in Artificial Intelligence LNAI*, pages 126–143. Cited in page 118

[Christensen and Haagh, 1996] Christensen, S. and Haagh, T. B. (1996). Design CPN - overview of CPN ML syntax. Technical report, University of Aarhus. Cited in page 93

[Colombetti et al., 2002] Colombetti, M., Fornara, N., and Verdiccchio, M. (2002). The role of institutions in multiagent systems. In *Atti del VII convegno dell'Associazione italiana per l'intelligenza artificiale (AI*IA 02)*. Cited in page 25

[Conte and Castelfranchi, 1993] Conte, R. and Castelfranchi, C. (1993). Norms as Mental Objects: From Normative Beliefs to Normative Goals. In *Procs. of MAAMAW'93*, Neuchatel, Switzerland. Cited in page 11

[Conte and Castelfranchi, 1995a] Conte, R. and Castelfranchi, C. (1995a). *Cognitive and social action*. UCL Press. Cited in page 13

[Conte and Castelfranchi, 1995b] Conte, R. and Castelfranchi, C. (1995b). Understanding the Functions of Norms in Social Groups through Simulation. In Gilbert, N. and Conte, R., editors, *Artificial Societies. The Computer Simulation of Social Life*, pages 252–267, London. UCL Press. Cited in page 11

[Cost et al., 2000] Cost, R. S., Chen, Y., Finin, T. W., Labrou, Y., and Peng, Y. (2000). Using colored petri nets for conversation modeling. In *Issues in Agent Communication*, pages 178–192, London, UK. Springer-Verlag. Cited in page 95

[Cranefield, 2005] Cranefield, S. (2005). A Rule Language for Modelling and Monitoring Social Expectations in Multi-Agent Systems. Technical Report 2005/01, University of Otago. Cited in page 20

[DEON, 2006] DEON (1991-2006). International Workshop in Deontic Logic in Computer Science. Cited in page 12

[Dignum, 1999] Dignum, F. (1999). Autonomous Agents with Norms. *Artificial Intelligence and Law*, 7(1):69–79. Cited in page 11

[Dignum, 2003] Dignum, V. (2003). *A model for organizational interaction: based on agents, founded in logic*. PhD thesis, Utrecht University. Cited in page 26, 27

[Elhag et al., 2000] Elhag, A., Breuker, J., and Brouwer, P. (2000). On the formal analysis of normative conflicts. *Information & Communications Technology Law*, 9(3):207–217. Cited in page 96

[Emerson and Halpern, 1986] Emerson, E. and Halpern, J. (1986). 'sometimes' and 'not never' revisited: on branching time versus linear time temporal logic. *Journal of the ACM*, 33:151–178. Cited in page 27

[Esteva, 2003] Esteva, M. (2003). *Electronic Institutions: from specification to development*. PhD thesis, Universitat Politecnica de Catalunya. Number 19 in IIIA Monograph Series. Cited in page 2, 23, 33, 44, 47, 48, 61, 62, 65, 82, 85

[Esteva et al., 2004] Esteva, M., Rosell, B., Rodríguez-Aguilar, J. A., and Arcos, J. L. (2004). AMELI: An agent-based middleware for electronic institutions. In *Proceedings of 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, pages 236–243. ACM. Cited in page 23, 24, 33, 101, 102, 103, 107

[Fitting, 1990] Fitting, M. (1990). *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, New York, U.S.A. Cited in page 52, 96

[Føllesdal and Hilpinen, 1981] Føllesdal, D. and Hilpinen, R. (1981). Deontic Logic: an introduction. In *Deontic Logic: Introductory and Systematic Readings*. D. Reidel. Cited in page 11

[Fornara et al., 2004] Fornara, N., Viganò, F., and Colombetti, M. (2004). A Communicative Act Library in the Context of Artificial Institutions. In *2nd European Workshop on Multi-Agent Systems*, pages 223–234, Barcelona. Cited in page 11, 19

[Frühwirth and Abdennadher, 2003] Frühwirth, T. and Abdennadher, S. (2003). *Essentials of Constraint Programming*. Springer Verlag. Cited in page 19

[Gaertner et al., 2007] Gaertner, D., García-Camino, A., Noriega, P., Rodríguez-Aguilar, J.-A., and Vasconcelos, W. (2007). Distributed Norm Management in Regulated Multi-agent Systems. In *Proceedings of 6th International Joint Conference on Autonomous Agents and Multiagent Systems. (AAMAS'07)*. Cited in page 9

[Gaertner et al., 2008] Gaertner, D., García-Camino, A., Noriega, P., Rodr'iguez-Aguilar, J. A., and Vasconcelos, W. (2008). Normative structures for regulating open multi-agent systems. *Autonomus Agents and Multi-Agent Systems*. (submitted). Cited in page 9

[Gaertner et al., 2006] Gaertner, D., Noriega, P., and Sierra, C. (2006). Extending the BDI architecture with commitments. In *Frontiers in Artificial Intelligence and Applications (to appear)*. IOS Press. Cited in page 118

[García-Camino, 2007] García-Camino, A. (2007). Ignoring, forcing and expecting concurrent events in electronic institutions. In *COIN III: Coordination, Organization, Institutions and Norms in Agent Systems. Revised Selected Papers from the 2007 Workshop Series*, volume 4870 of *Lecture Notes in Computer Science*, pages 15–26. Springer. Cited in page 8, 28

[García-Camino et al., 2005a] García-Camino, A., Noriega, P., and Rodríguez-Aguilar, J.-A. (2005a). Implementing Norms in Electronic Institutions. In *Proceedings of 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, pages 667–673, Utrecht, The Nederlands. Cited in page 7

[García-Camino et al., 2005b] García-Camino, A., Noriega, P., and Rodríguez-Aguilar, J.-A. (2005b). Implementing Norms in Electronic Institutions (Extended Abstract). In *3rd European Workshop on Multiagent Systems (EUMAS'05)*, Brussels, Belgium. Cited in page 8

[García-Camino et al., 2007a] García-Camino, A., Noriega, P., and Rodríguez-Aguilar, J. A. (2007a). An algorithm for conflict resolution in regulated compound activities. In *Engineering Societies in the Agents World VII*, volume 4457 of *Lecture Notes in Computer Science*, pages 193–208. Cited in page 9, 117

[García-Camino et al., 2006a] García-Camino, A., Rodríguez-Aguilar, J.-A., Sierra, C., and Vasconcelos, W. (2006a). A Distributed Architecture for Norm-Aware Agent Societies. In Baldoni, M. et al., editors, *Declarative Agent Languages and Technologies III*, volume 3904 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 89–105. Springer, Berlin Heidelberg. Cited in page 8, 48

[García-Camino et al., 2006b] García-Camino, A., Rodríguez-Aguilar, J.-A., Sierra, C., and Vasconcelos, W. (2006b). A Rule-based Approach to Norm-Oriented Programming of Electronic Institutions. *ACM SIGecom Exchanges*, 5(5):33–40. Cited in page 8

[García-Camino et al., 2006c] García-Camino, A., Rodríguez-Aguilar, J.-A., Sierra, C., and Vasconcelos, W. (2006c). Norm Oriented Programming of Electronic Institutions. In *Proceedings of 5th International Joint Conference on Autonomous Agents and Multiagent Systems. (AAMAS'06)*. Cited in page 8

[García-Camino et al., 2006d] García-Camino, A., Rodríguez-Aguilar, J.-A., Sierra, C., and Vasconcelos, W. (2006d). Norm-Oriented Programming of Electronic Institutions (Extended Abstract). In *Fourth European Workshop on Multi-Agent Systems (EUMAS'06)*, Lisbon, Portugal. Cited in page 8

[García-Camino et al., 2008] García-Camino, A., Rodríguez-Aguilar, J. A., Sierra, C., and Vasconcelos, W. (2008). Constraint rule-based programming of norms for electronic institutions. *Autonomus Agents and Multi-Agent Systems*. (In press). Cited in page 8

[García-Camino et al., 2007b] García-Camino, A., Rodríguez-Aguilar, J. A., Sierra, C., and Vasconcelos, W. W. (2007b). Norm-oriented programming of electronic institutions: A rule-based approach. In *Coordination, Organization, Institutions and Norms in agent systems II*, volume 4386 of *Lecture Notes in Computer Science*, pages 177–193. Springer-Verlag. Cited in page 8, 87

[García-Camino et al., 2007c] García-Camino, A., Rodríguez-Aguilar, J. A., and Vasconcelos, W. W. (2007c). A distributed architecture for norm management in multi-agent systems. In *COIN III: Coordination, Organization, Institutions and Norms in Agent Systems. Revised Selected Papers from the 2007 Workshop Series*, volume 4870 of *Lecture Notes in Computer Science*, pages 275–286. Springer. Cited in page 9

[Gelfond et al., 1991] Gelfond, M., Lifschitz, V., and Rabinov, A. (1991). What are the limitations of the Situation Calculus? In *Essays in Honor of Woody Bledsoe*, pages 167–179. Cited in page 77

[Giunchiglia and Serafini, 1994] Giunchiglia, F. and Serafini, L. (1994). Multilanguage hierarchical logics or: How we can do without modal logics. *Artificial Intelligence*, 65(1):29–70. Cited in page 113

[Gossling, 1996] Gossling, J. (1996). *The Java programming Language*. Reading. Addison-Wesley. Cited in page 37

[Hannoun et al., 2000] Hannoun, M., Boissier, O., Sichman, J. S., and Sayettat, C. (2000). MOISE: An organizational model for multi-agent systems. In Monard, M. C. and Sichman, J. S., editors, *Advances in Artificial Intelligence*, volume 1952 of *LNAI*. Springer-Verlag. Cited in page 24

[Harel, 1979] Harel, D. (1979). First-order dynamic logic. In *Lecture Notes in Computer Science*. Springer. Cited in page 11

[Hübner et al., 2005] Hübner, J. F., Sichman, J. S., and Boissier, O. (2005). S-MOISE+: a middleware for developing organized multi-agent systems. In *Proc. International Workshop on Organizations in Multi-Agent Systems: From Organizations to Organization-Oriented Programming (OOOP05)*, Utrecht, The Netherlands. Cited in page 25

[Huisjes, 1981] Huisjes, C. H. (1981). *Norms and Logic*. PhD thesis, University of Groningen. Cited in page 11

[Jaffar et al., 1998] Jaffar, J., Maher, M. J., Marriott, K., and Stuckey, P. J. (1998). The Semantics of Constraint Logic Programs. *Journal of Logic Programming*, 37(1-3):1–46. Cited in page 19

[Jennings et al., 1998] Jennings, N. R., Sycara, K., and Wooldridge, M. (1998). A roadmap of agent research and development. *Journal of Agents and Multi-Agents Systems*, 1:7–38. Cited in page 1

[Jensen, 1997] Jensen, K. (1997). *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Uses (Volume 1)*. Springer. Cited in page 86, 92

[Jones and Sergot, 1996] Jones, A. J. and Sergot, M. J. (1996). A formal characterisation of institutionalised power. *Journal of the Interest Group in Pure and Applied Logic*, 4(3):427–443. Cited in page 26

[Knuuttila, 1981] Knuuttila, S. (1981). The emergence of deontic logic in the fourteenth century. In Hilpinen, R., editor, *New Studies in Deontic Logic*, pages 225–248. D. Reidel Publishing Company. Cited in page 11

[Kollingbaum, 2005] Kollingbaum, M. J. (2005). *Norm-Governed Practical Reasoning Agents*. PhD thesis, Department of Computing Science, University of Aberdeen, United Kingdom. Cited in page 3, 18, 86, 112

[Kollingbaum and Norman, 2003a] Kollingbaum, M. J. and Norman, T. J. (2003a). NoA – A Normative Agent Architecture. In *Proceedings 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1465–1466, Acapulco, Mexico. AAAI Press. Cited in page 18

[Kollingbaum and Norman, 2003b] Kollingbaum, M. J. and Norman, T. J. (2003b). Norm Adoption in the NoA Agent Architecture. In *Proceedings 2nd International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS'03)*, Melbourne, Australia. ACM, U.S.A. Cited in page 18

[Kollingbaum et al., 2007a] Kollingbaum, M. J., Vasconcelos, W. W., García-Camino, A., and Norman, T. (2007a). Conflict resolution in norm-regulated environments via unification and constraints. In *Declarative Agent Languages and Technologies V*, volume 4897 of *Lecture Notes in Artificial Intelligence*, pages 158–174. Springer. Cited in page 9, 86, 97, 98

[Kollingbaum et al., 2007b] Kollingbaum, M. J., Vasconcelos, W. W., García-Camino, A., and Norman, T. (2007b). Managing conflict resolution in norm-regulated environments. In *Engineering Societies in the Agents World VIII*, volume (In press) of *Lecture Notes in Artificial Intelligence*. Springer. Cited in page 9, 99, 117

[Kramer and Mylopoulos, 1992] Kramer, B. and Mylopoulos, J. (1992). Knowledge Representation. In Shapiro, S. C., editor, *Encyclopedia of Artificial Intelligence*, volume 1, pages 743–759. John Wiley & Sons. Cited in page 88

[Lomuscio and Nute, 2004] Lomuscio, A. and Nute, D., editors (2004). *Proc. of the 7th Intl. Workshop on Deontic Logic in Computer Science (DEON'04)*, volume 3065 of *Lecture Notes in Artificial Intelligence*. Springer Verlag. Cited in page 11

[Lopes Cardoso and Oliveira, 2005] Lopes Cardoso, H. and Oliveira, E. (2005). Virtual Enterprise Normative Framework within Electronic Institutions. In *Engineering Societies in the Agents World V*, volume 3451 of *LNAI*, pages 14–32. Springer-Verlag. Cited in page 28

[Lopes Cardoso and Oliveira, 2007] Lopes Cardoso, H. and Oliveira, E. (2007). Electronic institutions for b2b: dynamic normative environments. *Artificial Intelligence and Law*. Cited in page 28

[López y López, 2003] López y López, F. (2003). *Social Power and Norms: Impact on agent behaviour*. PhD thesis, University of Southampton. Cited in page ix, 2, 15, 112

[López y López and Luck, 2004] López y López, F. and Luck, M. (2004). A Model of Normative Multi-Agent Systems and Dynamic Relationships. In *Regulated Agent-Based Social Systems*, volume 2934 of *LNAI*, pages 259–280. Springer-Verlag. Cited in page 15

[Meyer, 1987] Meyer, J.-J. C. (1987). A different approach to deontic logic: deontic logic viewed as a variant of dynamic logic. *Notre Dame Journal of Logic*, 29(1):109–136. Cited in page 11

[Michael et al., 2004] Michael, L., Parkes, D. C., and Pfeffer, A. (2004). Specifying and monitoring market mechanisms using rights and obligations. In *Proceedings AAMAS Workshop on Agent Mediated Electronic Commerce (AMEC VI)*, New York, USA. Cited in page 17

[Minsky, 2005] Minsky, N. H. (2005). Law Governed Interaction (LGI): A Distributed Coordination and Control Mechanism (An Introduction, and a Reference Manual). Technical report, Rutgers University. Cited in page 27

[Murata, 1989] Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580. Cited in page 96

[Noriega, 1997] Noriega, P. (1997). *Agent-Mediated Auctions: The Fishmarket Metaphor*. PhD thesis, Universitat Autonoma de Barcelona. Number 8 in IIIA Monograph Series. Cited in page 22, 62

[Okuyama et al., 2007] Okuyama, F. Y., Bordini, R. H., and da Rocha Costa, A. C. (2007). Spatially distributed normative objects. In *Coordination, Organizations, Institutions, and Norms in Agent Systems II*, volume 4386 of *Lecture Notes in Computer Science*, pages 133–146. Cited in page 117

[OMG, 2003] OMG (2003). Object constraint language specification 1.4. `http://www.omg.org`. Cited in page 19

[OMG, 2005] OMG (2005). Unified Modelling Language. `http://www.uml.org`. Cited in page 19

[Rodríguez-Aguilar, 2001] Rodríguez-Aguilar, J. A. (2001). *On the Design and Construction of Agent-mediated Electronic Institutions*. PhD thesis, Universitat Autonoma de Barcelona. Number 14 in IIIA Monograph Series. Cited in page 3, 22

[Ross, 1941] Ross, A. (1941). Imperatives and logic. *Theoria*, 7:53–71. Cited in page 12

[Sandia National Labs, 2006] Sandia National Labs (2006). *Jess. The Rule Engine for Java*. `http://www.jessrules.com`, viewed on 15 Mar 2006 at 17.50 GMT. Cited in page 34, 37, 44

[Sartor, 1992] Sartor, G. (1992). Normative conflicts in legal reasoning. *Artificial Intelligence and Law*, 1(2-3):209–235. Cited in page 3, 86

[Searle, 1995] Searle, J. (1995). *The Construction of Social Reality*. The Penguin Press, London. Cited in page 26, 28, 87, 116

[Sergot, 2001] Sergot, M. (2001). A Computational Theory of Normative Positions. *ACM Trans. Comput. Logic*, 2(4):581–622. Cited in page 12, 13

[Shoham and Tennenholtz, 1995] Shoham, Y. and Tennenholtz, M. (1995). On Social Laws for Artificial Agent Societies: Off-line Design. *Artificial Intelligence*, 73(1-2):231–252. Cited in page 11

[SICS, 2006] SICS (2006). *SICStus Prolog*. Swedish Institute of Computer Science. `http://www.sics.se/sicstus`, viewed on 10 Feb 2006 at 18.16 GMT+1. Cited in page 54

[Stratulat et al., 2001] Stratulat, T., Clérin-Debart, F., and Enjalbert, P. (2001). Norms and Time in Agent-based Systems. In *Proceedings 8th International Conference on AI & Law (ICAIL'01)*, pages 178 – 185, St. Louis, Missouri, USA. Cited in page 17

[Tsang, 1993] Tsang, E. P. K. (1993). *Foundations of Constraint Satisfaction*. Academic Press. Available at `http:/www.bracil.net/edward/FCS.html`. Cited in page 53

[Tuomela and Bonnevier-Tuomela, 1995] Tuomela, R. and Bonnevier-Tuomela, M. (1995). Norms and Agreement. *European Journal of Law, Philosophy and Computer Science*, 5:41–46. Cited in page 11

[Udupi and Singh, 2006] Udupi, Y. B. and Singh, M. P. (2006). Multiagent policy architecture for virtual bussiness organizations. In *Proceedings of the IEEE International Conference on Services Computing (SCC)*. Cited in page 28

[Vasconcelos et al., 2004] Vasconcelos, W. W., Robertson, D., Sierra, C., Esteva, M., Sabateri, J., and Wooldridge, M. (2004). Rapid Prototyping of Large Multi-Agent Systems through Logic Programming. *Annals of Mathematics and Artificial Intelligence*, 41(2–4):135–169. Cited in page 61

[Vázquez-Salceda et al., 2004] Vázquez-Salceda, J., Aldewereld, H., and Dignum, F. (2004). Implementing Norms in Multiagent Systems. In *Multiagent System Technologies: Second German Conference, MATES 2004*, volume 3187 of *LNAI*, pages 313–327, Erfurt, Germany. Springer-Verlag. Cited in page ix, 11, 14, 33

[von Wright, 1951] von Wright, G. H. (1951). Deontic logic. *Mind*, 60:1–15. Cited in page 3, 11

[von Wright, 1963] von Wright, G. H. (1963). *Norm and Action: A Logical Inquiry*. Routledge and Kegan Paul, London. Cited in page 11

[Walker and Wooldridge, 1995] Walker, A. and Wooldridge, M. J. (1995). Understanding the emergence of conventions in multi-agent systems. In *Proceedings International Joint Conference on Multi-Agent Systems (ICMAS)*, pages 384–389, San Francisco, USA. Cited in page 11