Normative Spaces for Open Electronic Institutional Environments

Marcos Antonio de Oliveira

A thesis submitted for the degree of

Doctor of Philosophy in Information Science

at the University of Otago, Dunedin,

New Zealand.

February 2013

Abstract

In the distributed multiagent systems discussed in this thesis, heterogeneous autonomous agents interoperate in order to achieve their goals. In such environments, agents can be embedded in diverse contexts and interact with agents of various types and behaviours. Mechanisms are needed for coordinating these multiagent interactions, and so far they have included tools for the support of conversation protocols and tools for the establishment and management of agent groups and electronic institutions. In this thesis, we describe our institutional environment approach, which includes the use of commitments and normative spaces. It is based on a metaphor in which agents may join an open system at any time, but they must obey regulations in order to maintain a suitable reputation that reflects its degree of cooperation with other agents in the group, and make them a more desired partner for others in a distributed organizational environment.

Coloured Petri Nets are used to formalize a workflow in the institutional environment defining a normative space that guides the agents during interactions in the conversation space. Model Checking is used to determine the change of states of the Commitment Objects that represent the Norms in the environment and Count-as Rules are used to separate lower level concerns of the System Agents to higher level concerns more related with the application domain where the system is built.

Acknowledgements

I would like to thank the following people who have helped and supported me with completing this thesis.

First and foremost, special thanks to my supervisors Martin Purvis and Stephen Cranefield, for their encouragement and support throughout this long journey. I am really thankful for all the time they have spent advising and helping me to finish this work.

I would like to thank my advisors Marcia Gottgtroy, for having encouraged me to pursue a research career, and Vasco Furtado for the valuable insights and critic offered during the last years of research in this work, and for allowing me to use the WikiCrimes system as a study case for this thesis.

Big thanks to my parents, Francisco and Maria, for having supported me and encouraged me to endeavour in my dreams and for them to bare the distance between New Zealand and Brazil. They have always made efforts to help me to pursue my goals and encouraged me to finish my PhD.

It would have been impossible for me to finish my PhD without the help and encouragement of my lovely wife Marjorie. She has always pushed me to finish my work, offering me love and patience when taking care of our son Lucas during the time I worked in this thesis.

Thanks to my son and best mate Lucas for trying to understand when daddy could not be around to play. The bed time talk was great for stress relief!

My personal thanks to my parents in law, Cicero and Fatima, and to my sisters in law Rebeca and Ive for all the words of advice, support and encouragement on the completion of my PhD.

Special thanks to my fellow mates from Otago, Mariusz Nowostawski, Ula Nowostawska, Tony Savarimuthu, Toktam Ebadi, Nasser Bagueri, Yasser Hammam, Sharmila Savarimuthu, Carl Lecther, Tony Moore, Da Deng, for all the words of advice and the great time together spent in Dunedin and all over New Zealand.

Great thanks goes to my friends form University of Fortaleza (UNIFOR), Eurico Vasconcelos, Carlos Caminha, Ricardo Perrone, Thiago Assunção, Carlos Gustavo, Adriano Melo, Mairon Belchior and many others that welcomed me in Brazil and offered me a good time and environment to work on my PhD.

I would like to thank also my workmates from Federal University of Ceará (UFC) for all the support and encouragement in finishing my work. The UFC has offered me the perfect environment to finish my PhD.

To my friends from Auckland, especially my mates Kelsen and Goreth, for always welcome me in their home whenever I needed to escape from the weather from Dunedin and during the ups and downs of this long journey.

Thanks as well go to my friends from Natal from the Federal University of Rio Grande do Norte (UFRN) class of 1992 and the following years. They were always ready to welcome me in Brazil and push me to finish my PhD.

Finally and most importantly, I would like to thank God for making this thesis possible!

Table of Contents

Preface ..... **xii**

1. Introduction **1**
1.1 Motivation 2
1.2 Openness Considerations 4
1.3 Research Justification and Contributions 5
1.4 Structure 7

2 State of the Art **8**
2.1 Modelling and Implementing Open Multiagent Systems 10
2.2 Institutions and Institutional Acts 19
2.3 Organizational Infrastructures for Open MAS 19
2.4 Communication in MAS 26
2.4.1 Representation of Interaction Protocols 28
2.5 Social Artefacts 33
2.6 Commitments between Agents for Social Control 35
2.7 Monitoring and Verification of Commitments 37
2.8 Counts-as Rules 40
2.9 Achieving Social Control in Normative Systems 41
2.10 Norm Enforcement and Regimentation 44
2.11 Summary 44

3 Institutional Environments **46**
3.1 The Idea behind Institutional Environments 46
3.2 Institutional Acts 50
3.3 Coloured Petri Nets to Model Conversations 53
3.4 Commitments for the Implementation of Institutional Norms 54
3.5 Reputation in the Institutional Environment 57
3.6 Specification of the Institutional Environment 59
3.6.1 Agent Roles and the Conversation Space 62

3.6.2 Commitments 64
3.6.3 Reputation Influence in the Institutional Environment 66
3.7 Workflows for the Institutional Environment 67
3.7.1 Institution Agent Workflow 67
3.7.2 Monitor Agent Workflow 73
3.7.3 Reputation Agent Workflow 75
3.7.4 Database Administrator Agent Workflow 79
3.7.5 Deputy Agent Workflow 77
3.8 The Normative Space 80
3.10 Freedom of Speech 82
3.11 Institutional Environment State 85
3.12 Engineering an Institutional Environment 86
3.13 Summary 87

4 The Technology Employed 88
4.1 Implementation Approach 88
4.2 The Otago Agent Platform (Opal) 89
4.2.1 Opal Architecture 90
4.3 Interoperability with the Normative Space 91
4.4 JFern Net Tool Box 95
4.4.1 Java Integration 95
4.4.2 Net Structure 95
4.4.3 Multiset 95
4.4.4 Place 96
4.4.5 Transition 96
4.4.6. Input Arc 96
4.4.7 Output Arc 96
4.4.8 Building Workflows 97
4.4.9 CPN Examples 98
4.5 The Model Checker 99
4.6 Summary 101

5 The WikiCrimes Institutional Environment 102
5.1 Introduction 102

5.2 Collaborative Systems 104
5.3 WikiCrimes: Motivation and Goals 105
5.4 WikiCrimes Institutional Environment 106
5.4.1 Roles present in the WikiCrimes Institutional Environment 107
5.4.2 Institutional Acts 108
5.4.3 WikiCrimes Institutional Environment Workflows 111
5.4.4 Commitments 118

6 WikiCrimes Institutional Environment Walkthroughs 122
6.1 External Agent Registration 122
6.1.1 Scenario Description 122
6.1.2 Activity Diagram for the Registration Process 124
6.1.3 Messages Involved in the Registration 124
6.1.4 High Level CPN 126
6.1.5 State Diagram of the Commitment Object 126
6.1.6 Model Checker Details 127
6.2 Register Criminal Fact 128
6.2.1 Scenario Description 128
6.2.2 Activity Diagram for the Criminal Fact Registration Process 130
6.2.3 Messages Involved in the Criminal Fact Registration 130
6.2.4 High Level CPN 133
6.2.5 State Diagram of the Commitment Object 134
6.2.6 Model Checker Details 134
6.3 Pre-Commitment to Confirm a Crime Acceptance 134
6.3.1 Scenario Description 134
6.3.2 Activity Diagram for the Pre-Commitment Acceptance Process 136
6.3.3 Messages Involved in the Commitment Proposal 136
6.3.4 High Level CPN 139
6.3.5 State Diagram of the Commitment Object 139
6.3.6 Model Checker Details 140
6.4. Confirm Crime Negatively 141
6.4.1 Scenario Description 141
6.4.2 Activity Diagram for the Negative Confirmation 143
6.4.3 Messages Involved in the Negative Confirmation 143

6.4.4 High Level CPN 145
6.4.5 State Diagram of the Commitment Object 145
6.4.6 Model Checker Details 146
6.5 Confirm a Crime Positively 148
6.5.1 Scenario Description 148
6.5.2 Activity Diagram for Positive Confirmation Process 148
6.5.3 Messages Involved in the Positive Confirmation 148
6.5.4. High Level CPN 151
6.5.5 State Diagram of the Commitment Object 152
6.5.6 Model Checker Details 152
6.6 Freedom of Speech 153
6.6.1 Scenario Description 153
6.6.2 Activity Diagram for the Free of Speech Process 154
6.6.3 Messages Involved in the Free Speech 154
6.6.4 High Level CPN 156

7 Conclusions 157
7.1 Software Engineering Issues 157
7.2 Discussion 159
7.2.1 Social Control 160
7.2.2 Model Checking and Count-as Rules 164
7.2.3 Social Commitments 165
7.3 Future Work 165
7.4 Contributions 166
7.4.1. Characteristics from the Developed Framework 167
7.5 Limitations 168

Bibliography 169

Appendix A – Coloured Petri Nets Annotations 181

List of Figures

1.1 Normative Space of the Institutional Environment 4

2.1 Common architecture for organization infrastructures in open MAS 20
2.2 Institutional Environment organizational infrastructure architecture 24
2.3 Example CPN 30

3.1 Institutional Environment and Institutional Acts 52
3.2 State machine of the lifecycle of commitments 55
3.3 Institution Agent Workflow 68
3.4 Monitor Agent Workflow 74
3.5 Reputation Agent Workflow 75
3.6 Database Administrator Agent Workflow 77
3.7 Deputy Agent Generic Workflow 78
3.8 Normative Space of the Institutional Environment 81
3.9 Freedom of Speech 82

4.1 An Opal System implemented by Opal Agents 90
4.2 Opal Management Console 91
4.3 FIPA ACL Messaging prepared in the Opal Agent Management Console …….. 91
4.4 Message Exchange facilities provided by Opal ……. 93
4.5 JFern Editor …… 97
4.6 Simple auction example with Java Inscriptions ……... 98
4.7 CPN Representing a player role and graphical representation of a subnet ……... 99

5.1 The WikiCrimes Collaborative System ……. 103
5.2 Institution Agent Workflow in JFern ……. 111
5.3 Monitor Agent Workflow in JFern …… 112
5.4 Reputation Agent Workflow in JFern …… 113
5.5 Database Administrator Agent in JFern ……. 114
5.6 Specific Behaviour of the Deputy Registered User Agent and Deputy Certifier

Entity Agent ……. 115
5.7 Deputy Registered User Workflow in JFern …….. 116
5.8 Deputy Certifier Entity Workflow in JFern ……... 116
5.9 Specific Behaviour of the Deputy Invited User Agent …….. 117
5.10 Deputy Invited User Workflow in JFern …….. 117
5.11 Deputy Browser User Workflow in JFern ……... 118

6.1 Activity Diagram for the External Agent Registration Process ……..... 124
6.2 High Level CPN demonstrating the conversation of agents involved in the process of registration …….. 126
6.3 State diagram for the social commitment acquired by the External Agent when it registers with the WikiCrimes Institutional Environment ……... 127
6.4 Activity Diagram for the Registration of a Criminal Fact ……. 130
6.5 High level CPN demonstrating the conversations of agents involved in the process of a criminal fact registration ……..... 133
6.6 State diagram for the pre-commitment created by the Monitor Agent for the agent that was invited to confirm a criminal fact ……. 134
6.7 Activity Diagram for the Commitment Proposal ……... 136
6.8 High level CPN demonstrating the conversations of agents involved in the process of commitment proposal ……. 139
6.9 State diagram updating a pre-commitment to a commitment for the agent that was invited to confirm a criminal fact and has accepted the proposal ……. 140
6.10 Activity Diagram for the negative confirmation of a Criminal Fact …… 143
6.11 High level CPN demonstrating the conversations of agents involved in the process of negative confirmation ……. 145
6.12 State diagram updating the commitment state due to a change in the truth value of a proposition present in its content formula ……. 146
6.13 Activity Diagram for the positive confirmation …….. 148
6.14 High level CPN demonstrating the conversations of agents involved in the process of negative confirmation ……. 151
6.15 State diagram updating the commitment state due to a change in the truth value of a proposition present in its content formula ……. 152
6.16 Activity Diagram for Freedom of Speech …… 154
6.17 High level CPN demonstrating the conversations of agents involved in the

process of free speech …….. 156

Preface

During the development of this PhD thesis the work it contemplates was progressively and partially published. In most of cases I was the main author of the publication (De Oliveira, Cranefield and Purvis, 2007), (De Oliveira and Purvis, 2008), (De Oliveira et al., 2009), (De Oliveira et al., 2004), (De Oliveira et al., 2004), (De Oliveira et al., 2005), (De Oliveira et al., 2008). I have as well applied my ideas on Institutional Environments to the WikiCrimes collaborative system (Furtado et al., 2010) and to the Opal Agent Platform (Nowostawski et al., 2006), where I was co-author and responsible for the adaptation of my work on institutional environments to WikiCrimes and Opal contexts. The work in this thesis has not been published elsewhere, except as noted above.

Marcos Antonio de Oliveira

Dunedin, New Zealand, February 2013.

Chapter 1

Introduction

This thesis focuses on the analysis, design and development of open Multiagent Systems (MAS). It is argued here that open multiagent systems can be designed and implemented as institutional environments where heterogeneous agents can participate by playing roles and interacting by means of institutional acts. To have that set up a model was designed and implemented in a framework where are employed technologies such as Coloured Petri Nets (CPN) (Jensen, 1992) (Cost et al., 2000), commitments (Fornara and Colombetti, 2002), reputation (Castelfranchi, 2000) (Conte and Paolucci, 2002), model checking (Cranefield and Winikoff, 2009) (Dragone, 2005) (Markey and Schnoebelen, 2003), object-oriented language, functional language, temporal logic formulas and an environment for development of FIPA-based agents and FIPA-ACL-based message exchange.

Those technologies are employed in this work to approach the following problems:

- Concurrent conversation management and representation (using CPN);
- Institutional norm representation and enforcement (using commitments);
- Implementation of social control, i.e. the imposition of constraints on the behaviour of agents that agree to become members of institution (using reputation).

This thesis explores various approaches to designing and implementing novel open MAS, where a common architecture for organizational infrastructures is employed (Hübner et al., 2010), see Figure 2.1. We claim to have developed a different architecture which leaves out middleware and organizational proxy agents (Hübner et al., 2010) and gives more consideration to the autonomy of the agents that take part in the open MAS.

Our main goal during this work was to give more autonomy to agents that join an open MAS allowing for conversations unpredicted at design time to take place in an Institutional Environment. This goal is one of the challenges that the implementation of open MAS brings to the research community: how to maintain good performance of the overall system without having to limit the behaviour of agents to a few interactions identified at design time. The techniques explored in this thesis were used to build a framework that offers mechanisms for implementing open MAS allowing unpredicted conversations and, by these means, allowing the occurrence of undesired behaviour as well. The intent is to be able to build open MASs that allow good and bad behaviour and at the same time to implement ways to reward or punish the agents according to their behaviour and the goals of the system. With that we claim that our approach affords the construction of more scalable agent-based systems that can operate in less constrained environments. This will facilitate the management of agents with a higher degree of autonomy, without forcing them to behave according to a limited number of interactions predicted at design time.

1.1. Motivation

The management of interoperation among agents is a complex task and requires robust techniques and methodologies to be applied in the development of reliable and open MAS. Generally this category of computational systems is used to model distributed scenarios where heterogeneous software entities, agents, interact to pursue particular or common goals. Even more demanding is the modelling and implementation of features that deliver an environment less constrained to those agents, allowing them to have the ultimate choice of obeying regulations or dealing with possible sanctions imposed by the MAS norms. After all, agents are autonomous entities and the biggest challenge is to have a coordination system where the they can be free to decide what to do but at the same time be encouraged or induced to obey the regulations of the artificial society in which they are entering.

Agent frameworks, such as JADE (Bellifemine, Poggi and Rimassa, 2000) and Opal (Nowostawski, Purvis and Cranefield, 2002), are developed on top of abstract architectures such as the FIPA Abstract Architecture (FIPA, 2002) that aim to offer standard means for message exchange between agents and, with that, ease the communication process in its lowest levels. Agent Communication Languages (ACL), such as the FIPA-ACL (FIPA, 2002) and KQML (Finin, Labrou and Mayfield, 1997), were introduced to define standard semantics for terms used during agent communication. Interaction protocols (IP) attempt to

balance the expressive power of such languages by defining patterns of behaviour that agents must follow to engage in a communicative interaction with other agents as an attempt at building MAS that interoperate in “predictable” ways.

The individual, local, cultural and social aspects of the communicator are significant and can often outweigh conventional concerns of software developers, such as the development of appropriate syntax, efficient coding systems, and suitable terminologies. What is necessary is an infrastructure where the agents can rely on mechanisms that will implement or offer efficient support for interoperability.

Our approach aims to use institutional concepts for modelling open MAS as social groups formed by heterogeneous agents where a set of regulations is established to manage the interactions among them. As in real life institutional environments, those agents assume predefined roles in the MAS and will be supervised by some authorities, defined as system agents. The system agents are part of the infrastructure built to manage commitments among agents, or groups of agents, and agents’ reputations in the institution.

From the assumption that artificial agents can be part of a structured society that follows rules, we present an approach for the use of institutional environments, commitments and reputation models to organise MAS in social structures based on roles. One major concern in our approach is to create an institutional environment where the degree of openness the member agents experience is part of the norms that regulate the MAS.

Concurrency is an intrinsic characteristic of open MAS. In our approach Coloured Petri Nets (Jensen, 1992) are used to model agent roles and, with that, all the interactive processes in the institution. CPNs bring to our approach well defined semantics which build upon true concurrency, hierarchical representations and an explicit description of both states and actions. Adding to that, CPNs have an elaborated set of computer tools supporting their drawing, simulation and formal analysis. The CPNs in our approach form a distributed environment that we named Normative Spaces, as seen in Figure 1.1 (the details and operations of the comments shown in the figure will be covered in more detail in Chapter 3).

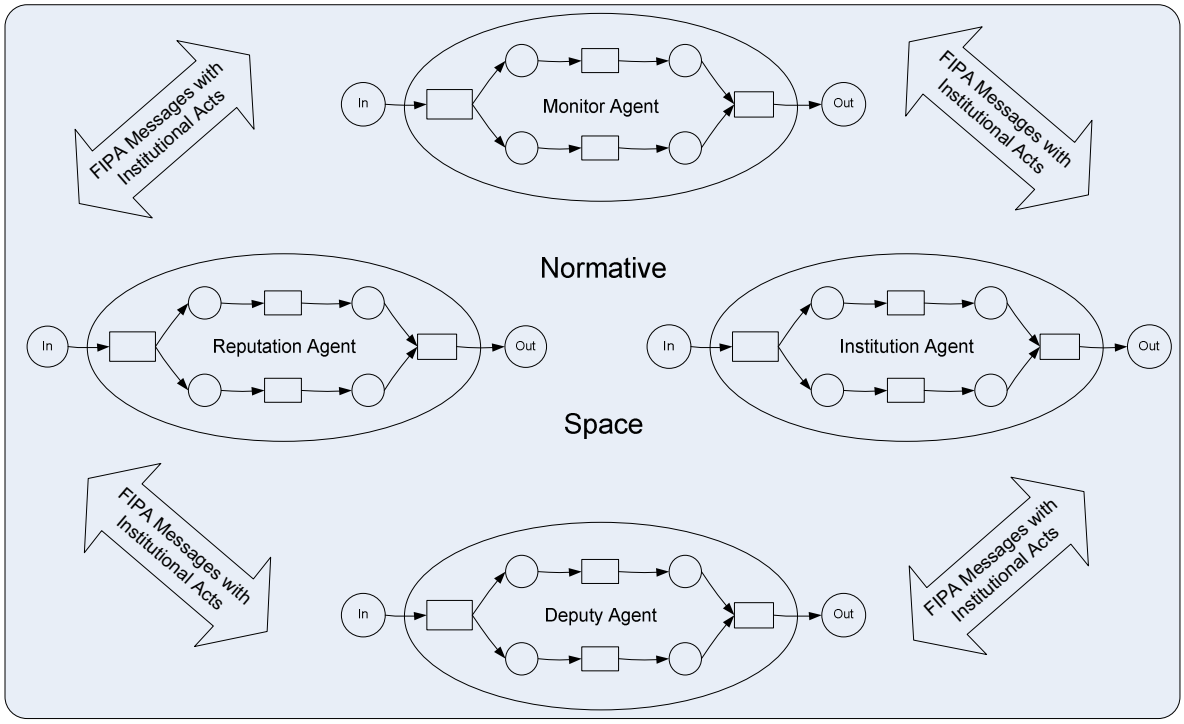


Figure 1.1: Normative Space of the Institutional Environment

In Chapter 2 related work is explored and the chapter highlights the choices made in prior research to ensure that MAS will behave as expected. In most prior research, interaction protocols are offered to external agents through Organizational Proxy Agents (Hübner et al., 2010) that constrain the behaviour of agents to patterns predicted at design time. In our work we offer support for predicted patterns of behaviour as well but, along with that, we offer the agent the choice to act as predicted or not and be rewarded or punished for that.

In Chapter 5 the specification of the WikiCrimes Institutional Environment is presented. It shows how we have implemented an open MAS using our framework. In Chapter 6 scenarios from the execution of the WikiCrimes Institutional Environment are pictured to demonstrate how the system behaved in predicted and unpredicted situations. This is to demonstrate that our framework allows for the construction of open MAS that are more concerned with offering an environment more suitable for managing agents with a high degree of autonomy.

1.2. Openness Considerations

How can an environment as heterogeneous as a human community be constrained? In different parts of the world different rules and norms are created so that the members of that group of people can feel a sense of security and order, and they can carry on with their lives

in a more predictable way. When a person needs some kind of service they know where to go, and if not they will ask another person or consult some kind of public catalogue. Once the person chooses to go to a place and make use of some sort of service he will make use of his past experiences and knowledge to carry on with actions, autonomously, and in a more or less standard way. Rules are there to be followed to organize the environment as well as to guide the members of a community so that they will have the option to follow them or not, depending on their goals in the environment and how they expect to be rewarded or punished by the environment.

1.3. Research Justification and Contributions

To manage autonomy in an open MAS is a challenge to the research community. In many cases the attempt to control autonomous agents has led to the development of systems with an excessive level of constraints to force the agents to behave according to what is expected. In our research we implement mechanisms to build open MAS with a variable level of constraints and suited to manage agents with a high degree of autonomy and able to develop conversations that are unpredicted at design time.

In this thesis a distributed model to implement open MAS is defined as an Institutional Environment. A set of technologies were used to build a framework suited to the development of Institutional Environments. It is claimed here that Institutional Environments are different from related work, described in Chapter 2, in the sense that they are essentially distributed and do not implement a middleware-like architecture.

The level of constraints in the environment is flexible – the agents that join the Institutional Environment can break or follow the norms of the system, represented as commitments between agents, and be rewarded or punished by adding or subtracting reputation points. The reputation of the agents is used to persuade them to follow the rules of the environment but at the same time leaves them free to act as they decide. We believe that the use of commitments associated with reputation can implement social order in the Institutional Environment where the constraints of the system are managed by the use of these two concepts.

Institutional Environments bring the following benefits:

- External agents can participate in both audited and non-audited interactions, and our institutional environments provide auditing and reputation management services so that the developers of external agents do not need to write code for that.
- In our approach we implement elements that aim to make the institutional environment more distributed. This is different from the majority of approaches present in the current research where there is a great use of middleware to provide a controlled environment for the implementation of organizational infrastructures. We provide a way to develop institutional environments where the external agents do not need to worry about how the services provided by the system agents from the environment are implemented.
- The institutional framework built supports asynchronous agent interaction via the FIPA ACL messages (FIPA, 2002), to all the agents registered in the institutional environment. This allows for more flexibility when implementing conversations among agents.
- Agents present in the institutional environment implement behaviours in the form of workflows defined as Coloured Petri Nets. This allows for the modelling and implementation of conversations with a tool made to handle concurrent processes.
- The organizational control of the institutional environment is distributed in the agents that are part of it. With that we aim to have a more scalable environment suitable to manage concurrent and asynchronous conversations.
- In our case study, in Chapter 5 and 6, the gains of developing a Collaborative System, such as WikiCrimes (www.wikicrimes.org), as an Institutional Environment can be observed. That opens a path for further research in that area together with the implementation of social control in that kind of system.
- With the use of Clojure Functional Language (Halloway, 2009) for the annotations of the CPNs, we have found more flexibility when representing workflows in the Institutional Environment.
- In our approach we had the concern to separate system level concepts from the domain level concepts. Applying counts-as rules to manipulate low level concerns of the Institutional Environment addresses issues that would influence higher level concerns of the agents involved in the Institutional Environment.
- We use commitment objects for the implementation of norms in an Institutional Environment. This choice fits well with our proposal to implement norms that

regulate the environment and norms that regulate agreements unpredicted at design time among agents present in the institutional environment.

- Our approach uses a model checker for the manipulation of a history of facts, relevant for the change of state of the commitment objects. These commitments represent the social norms in our Institutional Environments. The model checker verifies temporal propositions used to define the condition and content of commitments, allowing for the management of the temporal characteristics of these commitments' attributes.

1.4. Structure

The following chapters of this thesis are organized as follows:

Chapter 2 – A description of the state of the art involving the research related to ours and the technology employed in our work. In this chapter we briefly discuss the similarities and difference of our approach with those we surveyed.

Chapter 3 – We introduce the definition of institutional environments and draw a scheme for the development of systems using this model for the implementation of open MAS.

Chapter 4 – We describe in this chapter all the technology employed for the development of institutional environments, and justify their use.

Chapter 5 – This chapter describes the implementation of a prototype of institutional environment inspired by the WikiCrimes Collaborative System. We explain the nature of the WikiCrimes System highlighting its characteristics as an open MAS and its suitability as a scenario to develop our proof-of-concept.

Chapter 6 – We describe scenarios of runs executed in the prototype of the WikiCrimes institutional environment, exploring the messages exchanged among the agents involved in the scenarios, how the system agents manage the commitment objects, the reputation of the registered agents and the free speech among registered agents in the Institutional Environment.

Chapter 7 – We draw conclusions taken from our research and indicate future directions for it.

Chapter 2

State of the Art

In this chapter, related work is presented and, when pertinent, comments on details of our approach to the modelling and implementation of open MAS are made.

When modelling and implementing an open MAS that allows heterogeneous agents to join the system and perform tasks we use the abstraction of institutional environments and normative spaces, as introduced in the first chapter of this thesis. As the agents that join the institution are heterogeneous the necessity of the insertion of social norms in the system becomes evident, because these agents have their own goals and interests. These agents are built by developers that embed in them their own strategies to explore the open MAS and take from it as much as they believe they can.

Similarly to human societies, open MAS have to deal with different types of agents: the ones that behave as expected; the ones that want to take as much as they can from the system; and the ones that just want to break the rules or exploit the system by performing undesired behaviour and causing problems to other members of the agent society, or even to the agent society itself.

Norms are introduced to balance the functioning of the system and introduce an adaptable mechanism of control in the environment, adequate to the application domain that is modelled by the open MAS. With that mechanism, rules are introduced in the system. These rules try to drive the behaviour of the agents in a way that the goals of the system as a whole are observed leaving the autonomous agents in the system aware of the rules,

rewards and punishments they might have when attempting to achieve their own goals (Aldewereld et al., 2006).

The degree of openness observed in normative systems is variable. Norms define what is socially acceptable and unacceptable in the system and at the same time influence the agents to behave in a desired way, much as legal frameworks are developed to guide humans in the real world. It is important to have the mechanisms for enforcement of norms over the autonomous agents to grant a sense of order to open MAS, and for the degree of openness to be as variable as the domain modelled needs or the system designer desires.

Interaction and coordination are identified as major concerns when designing and deploying MAS, giving a distinct approach towards the modelling and design of distributed intelligent systems (DeLoach, 2002), (Zambonelli, Jennings and Wooldridge, 2003). Software engineering agent-oriented methodologies, such as Gaia (Wooldridge, Jennings and Kinny, 2000), have been developed to observe the interaction between agents as a critical design aspect when building MAS.

The system organization as well influences the design of an open MAS (DeLoach, 2002), (Zambonelli, Jennings and Wooldridge, 2001). A social setting is realized in the form of an environment where agents play roles and interact with each other pursuing individual or common goals. The organization has a defined structure composed of elements that are used to define and enforce norms to manage the interoperations among agents. Norms are associated with roles that agents assume in the system upon registration and will guide the agent behaviour in the system.

The operational use of norms in institutional environments is directly related to the context the agents are inserted into and it is defined through ontologies (Grossi et al., 2006) and (De Oliveira et al., 2005). Our approach is to use Coloured Petri Nets (CPN) (Jensen, 1992) to represent normative spaces that will guide agents throughout their useful existence in the institutional environment (De Oliveira et al., 2004), (De Oliveira et al., 2004), (De Oliveira, Cranefield and Purvis, 2007), (De Oliveira and Purvis, 2008), (De Oliveira et al., 2008), (De Oliveira et al., 2009), (Purvis et al., 2006), and (Nowostawski et al., 2006). In the following sections of this chapter a selection of related work is addressed. Similarities and differences from those studies are compared with the one proposed in this thesis. More details on the technical devices employed in this thesis are presented in Chapter 4.

2.1. Modelling and Implementing Open Multiagent Systems

Multiagent systems (MAS) are systems comprising multiple agents with different degrees of autonomy and which interact to achieve individual or common goals. In MAS because of the characteristics of the individual agents some types of interactions are necessary, as highlighted in Jennings, Sycara and Wooldridge (1998):

- Cooperation – collaboration for the achievement of a common objective;
- Coordination – management of the interactions to avoid behaviour that would deviate the functioning of the MAS to undesired states;
- Negotiation – the reach of agreements among the participating agents.

Such types of interaction are very characteristic of MAS due to the autonomy their agent members own. Problems where there are distributed sources of information, control, resources and knowledge are very much suited to being treated by MAS (Wooldridge and Jennings, 1999). Reusability of agent architectures and interaction protocols suitable to different application domains are other features inherent to agents that can make them more portable (Jennings, 2000).

Methodologies and models for the development of MAS have to deal with its inherent distribution, as well as with the characteristics of the agents that compose it. Autonomy and pro-activity are certainly difficult to manage, especially if the MAS is open to the participation of heterogeneous agents developed by third parties with their own self-interests and goals. Therefore, mechanisms to make the agents behave as expected by the MAS are necessary and they can allow only acceptable behaviour, and take out part of the agent´s autonomy, or adopt punishment and rewarding strategies so the agent is tagged with a label for good or bad reputation in the MAS.

We believe that a truly open MAS should allow for the participation of any kind of self-interested agents. Who knows and can predict completely what the self interests of the agents are? It is a big simplification to assume that they will always behave badly and force them to behave only in predicted ways. This might be very safe on one hand but on the other it limits the very intrinsic characteristic of the agent, its autonomy. Completely constrained systems can be implemented by other paradigms, such as the object-oriented, not the multiagent one.

Joining a society might require accepting limits on your autonomy, but the autonomy is still there to break the rules, and in some cases that might be necessary to obtain an instant punishment but maybe a bigger reward some time later. That scenario is present in the model of organization of human societies. In a soccer game, for example, you should not push the other players, or even commit a penalty, but you still can do it. The player might lose the penalty kick and the one that committed the penalty would get a yellow card but has maximized his or her gains.

Therefore, we also believe that an open MAS should implement mechanisms to deal with the performance of unauthorized actions, or unpredicted acts. In an attempt to model open MAS using the human model to organize their societies, in our approach we introduce mechanisms that allow for the classification of different kinds of agents in the open MAS and allow as well for the use of any conversation strategy the agents need or want to use. Who knows what could be the marketing strategy of an open MAS that implements an e-commerce company. For example, they might need to attract the most diversified profiles, which might include even the ones that do not always follow the rules or act predictably. This feature is not supported by other approaches such as Esteva (2003), Minsky and Ungureanu (2000), Hübner, Sichman and Boissier (2007) and Hübner et al. (2010). There is always a great worry about making sure that agents follow the rules and that agent interaction be more predictable so that it is more feasible to implement MAS.

The concern of not restricting completely the autonomy of agents in open MAS was discussed by Brafman (1996) and Criado, Argente and Botti (2011). Brafman defines Partially Controlled MAS (PAC) and visualizes MAS with two types of agents: *controllable* agents that are controlled directly by the system designer; and *uncontrollable* agents that are not under the designer´s direct control, and are allowed to exhibit deviating behaviour. Therefore in PAC (Brafman, 1996), controllable agents supervise the behaviour of uncontrollable ones enforcing desired behaviour. Criado, Argente and Botti (2011) describe THOMAS (Methods, Techniques and Tools for Open Multi-Agent Systems), a framework that does not have a mediator layer. THOMAS is being developed with the intent to allow for the agents to interact freely under a conceptual framework that categorizes types of norms that are enforced in the open MAS.

A number of application domains are suitable for the inclusion of self-interested software entities that reflect the goals of their developers or their own strategies of behaviours. There

are a great number of companies in the Internet looking for a variety of consumers demands models that implement open MAS. Social networks such as Facebook (www.facebook.com), Orkut (www.orkut.com) and MySpace (www.myspace.com), with their vast number of participants, are a potential market for companies from a wide variety of areas. The same applies to virtual environments such as Second Life (www.secondlife.com) (Rymaszewski, 2007), which is a perfect place to introduce self-interested agents that sell products or ideas. Application domains that deal with ubiquitous computing (Poslad, 2009) where software runs on a great variety of gadgets, such as mobile phones, are another example of very distributed environments that call for models that manage open MAS.

The complexity of the design and implementation of MAS has led to the idea of modelling them as organizations of agents, as introduced in early studies from Gasser, Braganza and Herman (1987), Pattison, Corkill and Lesser (1987), Corkill and Lesser (1983) and Werner (1987). Organizational approaches identifies roles that agents can assume in the MAS, and see the whole system as a unique structure where the agents are participants Esteva (2003).

The other way to implement an MAS is having an agent-centred view of it. Therefore, when modelling the systems the agents are identified and built with a specific purpose, and predefined conversations are set so they can achieve their goals (Esteva, 2003). In this case, there is a great tendency to make assumptions of benevolence and cooperation between the agents, which is not guaranteed in an open MAS. Agent-centered approaches are more suited to small systems that are not open MAS (Rodríguez-Aguilar, 2001), (Esteva et al., 2001). That is the case because in an open MAS it is necessary to deal with the heterogeneity of the agents, which are developed by people different from those who developed the open MAS. Therefore, the necessity is justified of implementing mechanisms to manage unpredictable behaviour of agents or agents that do not always play according to the rules of the game.

A number of methodologies have been proposed for the design and development of MAS (Iglesias, Garijo and Gonzalez, 1999), (Wooldridge and Ciancarini, 2001). Most of these methodologies are extensions of existing methodologies in other fields (mainly object-oriented programming and knowledge engineering) to include relevant aspects of agents, these fields that are not sufficient to cover all the aspects needed for the design of MAS (Esteva, 2003). Therefore, other approaches like the organization-oriented ones became

more relevant and more adapted to the modelling and implementation of MAS, especially when we endeavour to support the openness characteristics of this paradigm that have become more and more evident recently.

Now we discuss briefly some of the methodologies proposed to design and develop MAS:

- GAIA is a methodology made to model MAS as organizations where agents play roles to the level where engineers can implement it. The major criticism of this methodology is that the organizational structure is not explicitly defined, and that because it assumes that the agents comprising the MAS have a common goal it is not suited to model open MAS (Esteva, 2003).

- Roadmap (Juan, Pierce and Sterling, 2002) extends Gaia in modelling complex open systems. A new interaction model that allows the specification of protocols in AUML (Bauer, Müller and Odell, 2001) is added. The use-case model from UML (Martin and Kendall, 2000) is adapted to MAS requirements gathering.

- Ferber and Gutknetch (1998) elaborated an organizational approach based in the concepts of group, role and agent. Groups can be created with allowed roles, and role-to-role protocols are specified. The agents as well can participate in more than one group. The MadKit platform (Gutknecht and Ferber, 2000) was developed to validate the approach.

- The Tropos Methodology (Giunchiglia, Mylopoulos and Perini, 2002) defines activities for all the life cycle of development, from requirements to the implementation. The construction of MAS is divided in five phases: early requirements, later requirements, architectural design, detailed design and implementation. AUML (Bauer, Müller and Odell, 2001) is used to define protocols, and at the implementation phase agent skeletons based on the agent BDI architecture (Rao and Georgeff, 1995) and the model created are generated to be extended by the implementation phase. These skeletons are suitable for the JACK platform (Howden et al., 2001).

- The Prometheus Methodology (Padgham and Winikoff, 2002) consists of three phases: system specification, architectural design and detailed design. Functionalities are defined in the specification phase. Actions, perceptions and data

associated to functionalities are defined. Functionalities are assigned to agents in the architectural design phase and interaction protocols are defined in AUML (Bauer, Müller and Odell, 2001). Using a BDI architecture agents' internal structures are generated in the detailed design phase. A software tool was developed to support the design and generation of agent skeletons for the JACK platform (Howden et al., 2001).

- HARMONIA (Vàzquez-Scaleda and Dignum, 2003) is a framework for agent organizations. This framework takes into consideration institution objectives and their fulfillment, as well as norms that are defined on a level more abstract than institution structures and procedures (Dignum, 2002). Deontic logic is used to define abstract norms that are translated into concrete norms. The concrete norms are defined as rules that either specify the part of procedures that enforce them, or specify the events and triggers that fires a violation.

Analysing the methodologies described in this chapter, we can identify that their fundamental aspect is the identification of roles and their relationships, except for Prometheus in which functionalities are associated with the agents, which might be seen as a categorization of agents as well as the identification of agent roles in a MAS. Tropos, Roadmap and Prometheus specify interaction protocols using AUML. In the work of Esteva et al. (Esteva, 2003) (Esteva, de la Cruz and Sierra, 2002) they use Finite State Machines to define the interaction protocols, and we use Coloured Petri Nets due to their characteristics that to our view are more suited to model interaction protocols (this is explored more in Section 2.4). When the Institutional Environment approach is used in production software development, then it would be recommended that a principled agent-based methodology be used.

The difference in the methodologies is exactly what kind of system that each of them can model and produce. The objective of both Tropos and Prometheus is to obtain a set of agents that behave to achieve different goals of the system and that interact when necessary by following the defined protocols. They represent an example of designing and developing a MAS taking an agent-centred approach. There is no notion of organization; they commit to a concrete agent architecture and there is an assumption of agents' benevolence and cooperation among the resulting agents. Although they can be useful for the design of

closed systems, and they offer tools that generate agent skeletons thus reducing the development time of the system, they are not appropriate for open systems (Esteva, 2003).

The other methodologies listed and the electronic institutions defined in Esteva (2003), as well as ours, do not commit to any agent architecture, assuming that the system will be populated by heterogeneous agents. We believe that the organizational approach is more suited to the specification of open MAS, similar to Esteva (2003) and Ferber and Gutknetch (1998). In our approach we define system agents that take care of the organizational structure of the MAS and employ mechanisms to entice external agents to cooperate with the environment, without compromising the unique characteristic of heterogeneous agents, which is their autonomy. The designer of our kind of open MAS has the freedom to decide the degree of freedom the agents have in the systems by making the system more or less constrained according to the necessity imposed by the application domain.

In many previous works researchers have tried to adapt object-oriented methodologies to design MAS. As the Unified Modelling Language (UML) is a well-known standard for modelling object-oriented systems, many attempts to adapt its use in the modelling of MAS were done, for example, the work of Odell, Parunak and Bauer (2000), Parunak and Odell (2002) and Bauer, Miller and Odell (2001). AUML, like UML, is a language and not a methodology; it is a well-built extension of UML to the agent paradigm. Extensions of UML diagrams were proposed by different methodologies. In ROADMAP, for instance, the use of use cases is proposed to capture the requirements of the system and AUML interaction diagrams are used to represent interactions among agents.

Social design and agent design is differentiated in Dellarocas and Klein (1999). It is necessary to design the rules of the environment the participants of the environment and overall assume that agents are autonomous, proactive and self-interested, or in other words they will not behave only as expected and follow always the rules of the organization. In Dellarocas and Klein (1999) they define the following services: socialization, notary and expectation handling. The socialization service produces contracts (Dellarocas, 2000) where it is defined for the agents registered with the organization their rights and capabilities. The notary service verifies if the agent is following the rules of the system and the contract, and the exception handling service manages known exception types through sentinel agents.

We do believe that the work from Dellarocas (Dellarocas and Klein, 1999) (Dellarocas, 2000) is a good approach, and one of the problems they confront is how to represent the contracts that regulate the participation of the agents in the society, or the open MAS. In our view, commitments are a more suitable formalism to define the norms of the system because they are more punctual and simple to represent. Further, a set of commitments that the agents assume with the open MAS can be seen as a contract that the agents assume with the society.

Other point worth mentioning is to identify that it is necessary to explicitly enforce the systems's rules; otherwise the agents will not follow them. We use reputation as a concept that should be used to manage enforcement of the rules in the open MAS. As will be explained in the following chapters, the commitments the agents have with the society can be broken and their reputation in the system will be updated according to the observation of these commitments by the agents that should follow them. In our view, this allows for a better representation of a true open MAS where participating agents exercise all the autonomy that they possess. In our approach, we attempt to manage environments composed of truly autonomous agents: after all this is one of the big aims of the research community, to develop really autonomous agents. Therefore, our aim is to deliver an environment that can fit the necessity of the agent society designer; if he/she needs to have a very constrained environment he/she will be able to do so, and if he/she needs to lower the level of constraints it will be possible to do so, because we offer mechanisms to implement such versatility.

The work of Pablo Noriega (Noriega, 1997), Juan Antonio Rodrìguez-Aguilar (Rodríguez-Aguilar, 2001) and Marc Esteva (Esteva, 2003) has produced the definition of Electronic Institutions. In (Noriega, 1997) the basic concepts of electronic institutions are introduced through a metaphor of a trade institution (a fish market). In (Rodríguez-Aguilar, 2001) the concept of electronic institutions is extended, the fish market is implemented and further work is proposed to make the architecture extensible to develop any institution of agents. In Esteva (2003) the concepts are adjusted and a textual specification language is presented, a specification and verification tool was developed, and a generic infrastructure is made available to deploy institutions in different application domains.

In (Esteva, 2003) there are defined components of what Hübner et al. (Hübner et al., 2010) defines as Organizational Infrastructures (OI) (see next section). The main tools defined by

Esteva are Islander (Esteva, de la Cruz and Sierra, 2002) and Ameli (Esteva et al., 2004). The first offers a graphical editor to model electronic institutions as sets of scenes where agents meet and engage in interaction protocols; the second is an agent-based middleware for electronic institutions where an environment is implemented so that external agents can be represented in the middleware by a special interface agent, the governors, and together these make the external agents follow the regulations of the electronic institutions. In Arcos et al. (2005) they define a set of tools, including Islander and Ameli, that together comprise the Electronic Institution Development Environment (EIDE), and claim that it is suited to modelling and implementing open MAS.

García-Camino et al. (2009) define an approach to specify and manage the normative positions of agents (permissions, prohibitions and obligations). This is a rule-based formalism that includes constraints and is used to provide explicitly norms in Electronic Institutions (Esteva, 2003). This approach is an attempt at implementing conversations that were not predicted at design time in Electronic Institutions. In contrast, we use potentially audited commitments (Fornara et al., 2008) and reputation management to conduct the behaviour of agents in our Institutional Environment.

One of the main goals of our approach is to define and maintain an environment where agents will interact observing a set of rules or norms but will not necessarily or compulsorily have their actions restricted via some kind of interface agent, for example, the governors in the Electronic Institution Development Environment (EIDE) (Arcos et al., 2005), or the controllers in Law-governed Interaction (LGI) (Minsky and Ungureanu, 2000), where both implement a set of interface agents and a middleware.

In Dignum, Aldewereld and Dignum (2011) a comparison is made among methodologies to model and implement MAS as organizations. In their work it is argued that characteristics identified in the application domain lead to the choice of the most suitable method. A set of attributes is defined to guide that choice: complexity, uncertainty, environment, emergence, goal autonomy and control. We have an external organizational representation but we allow for interactions unpredicted at design time to take place in our organizational design. In our organizational design the external agents will have access to the institutional environment through an organizational proxy agent or interface agent but unpredicted interactions can happen, which differs from prior approaches such as MOISE+ (Hübner, Sichman and Boissier, 2007) and AMELI (Esteva et al., 2004). In Dignum, Aldewereld and Dignum

(2011) it is argued that the different organizational designs are applied to different contexts and they are suitable for certain conditions, goals and limitations.

Our approach is to offer mechanisms that will induce the agents to play a cooperative role in the agent society, but ultimately the choice of cooperating or not cooperating will be theirs. That choice might cause the employment of sanctions by the system agents on non-cooperative agents, and with that we will be able to use a model closer to the human way of organizing their societies. Legal systems developed to guide and regulate human behaviour in organizations such as universities, traffic, companies and many other groups or societies provide evidence that humans apply sanctions to establish and maintain social order in their societies.

Therefore, among the empirical concerns of electronic institutions (Noriega, 2006), we are more concerned with the choice the agents will have to make when deciding whether to join the institution of agents by evaluating the degree of freedom of speech that the institution will allow. By that we mean that as autonomous agents ourselves, we have individual goals and beliefs and those are in the autonomous artificial agents that we build. They are there because the agent-oriented paradigm is mirrored in the human society model and for a completely constrained environment the object-oriented paradigm is already there to be used.

Such mechanisms as governors (Arcos et al., 2005) and controllers (Minsky and Ungureanu, 2000), to our view, may compromise the agents' autonomy too much. We want to *influence* agents to behave according to the rules of the environment. That is the motivation of our work, to develop an environment where agents are influenced to cooperate and follow a predefined set of rules. That environment is organized based on institutional concepts and the assignment of roles to joining agents occurs as well. Those concepts create an artificial environment similar to real-world institutions where people can join to obtain and/or offer access to services.

Autonomous agent designers aim to copy human reasoning and/or strategies in their agents when they interact with others and decide their course of action in the electronic environment. Interaction protocols are there to be used in accordance with the speech acts, institutional actions, or illocutions identified in the dialogs executed by agents. Those interaction protocols help to predict actions and model the conversation space before it actually happens, including the possible breach of expected courses of action by agents in

the electronic environment. Authorities in the institutions are available to audit interactions and observe rules in the society.

The rigid control of agent interoperation may or may not grant rigid security for MAS but it definitely compromises the degree of agents' autonomy. If agents are only allowed to follow the rules, part of the intrinsic characteristics of the multi-agent model might be overlooked and their capability to deal with real-world situations diminished. We therefore advocate for a model where the openness of the MAS is taken into consideration when defining the norms of the institutional environment.

2.2. Institutions and Institutional Acts

Institutions (Colombetti, Fornara and Verdicchio, 2002) (Esteva et al., 2001) are software organizations where agents assume pre-defined roles to interact in an environment that have rules of conduct defined. Those rules can be defined, for example, as norms (Dignum, 2002), contracts (Dellarocas, 2000), policies (Elio and Haddadi, 1999) or commitments (Colombetti, Fornara and Verdicchio, 2004) (Singh, 1999). In the institutions are defined ontologies to determine the terms to be used in the conversations that take place in the institution.

Institutional acts (Colombetti and Verdicchio, 2002) (Colombetti, Fornara and Verdicchio, 2004) are terms identified in the institutional environment that when used bring about institutional effects changing the state of the institutional environment. They are used in interaction protocols that lead to conversations that will possibly change the current state of the institution.

The technology applied in the development of electronic institutions have followed the pattern of construction of middleware-like environments that support the services offered by the institution, better explored in the next section.

2.3. Organizational Infrastructures for Open MAS

Work on the development of techniques for the implementation of MAS has concentrated lately on the conception of Organizational Infrastructures (OI) (Hübner et al., 2010). There is a clear intention to implement and manage open MAS through the use of such

techniques, but there is still some work to be done especially on the management of the openness.

Software agents are autonomous entities that behave pro-actively to adapt to the environment where they are found. Organizational infrastructures (OIs) are put in place to organize and regulate the behaviour of the agents in the environment. Concepts such as trust, reputation, coordination, norms, heterogeneity and openness must be taken into consideration when implementing open MAS and consequently must be present in the OIs.

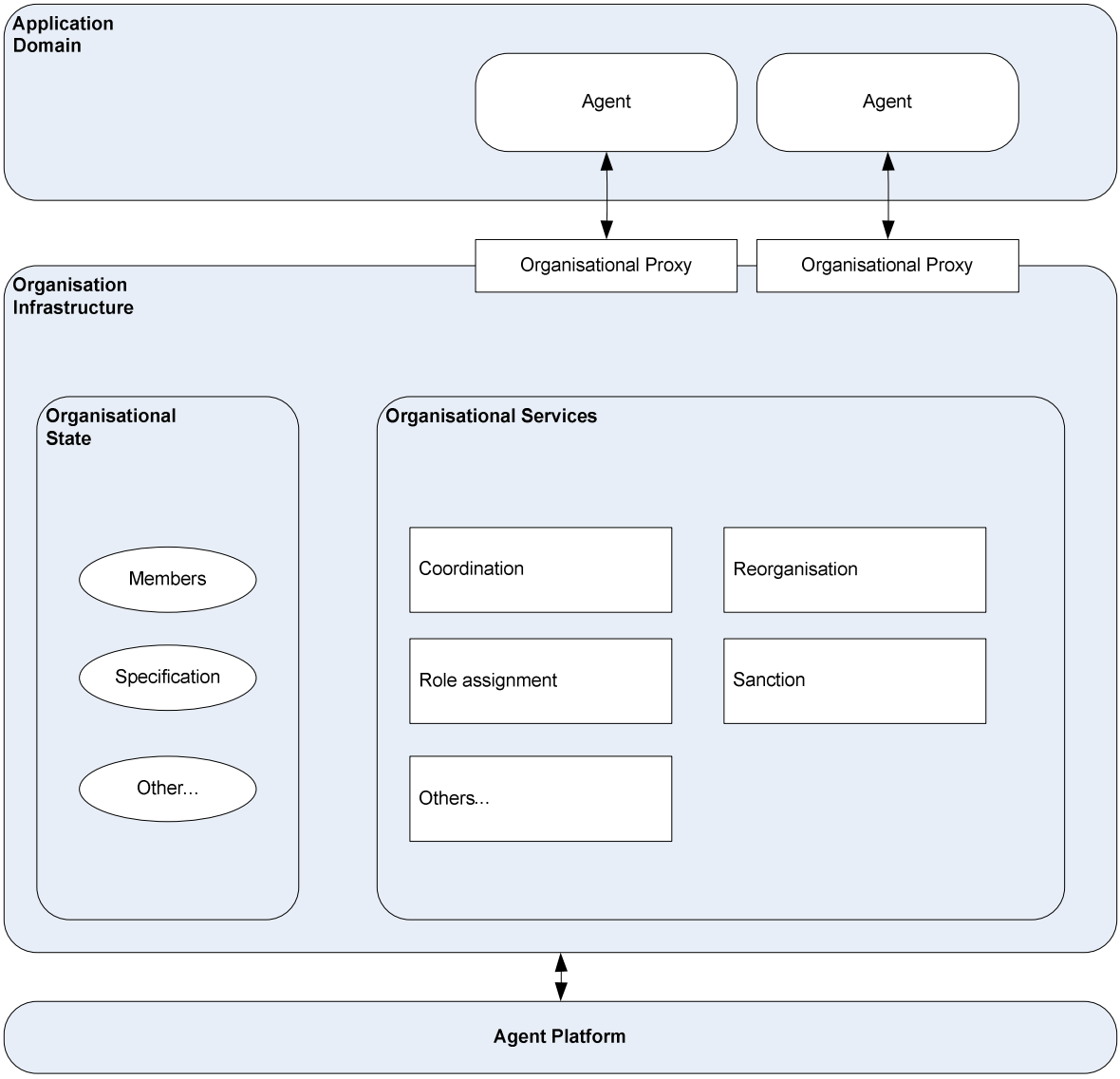


Figure 2.1. Common architecture for organization infrastructures in open MAS redrawn from (Hübner et al., 2010)

Hübner et al. (2010) defined a common architecture for organizational infrastructures (redrawn in Figure 2.1). The architecture is composed of services and agents that take care

of the functioning of the OI. The agents shown in the “Application Domain” of Figure 2.1 are external agents that interact with the organisational middleware that is in charge of the management of the services offered by the OI and its internal state. Usually the OI middleware imposed rigid restrictions on the behaviour of the external agents in connection with their interaction with the organisation.

Our approach is to distribute the management usually imposed by the centralized middleware in system agents that have predefined management roles and together manage the whole institutional environment. Furthermore, we introduce “freedom of speech” to the application agents so that they can have unmonitored interactions regarding any subject they want, thus making the management of the openness of the environment softer than the usual control imposed by middleware, where the agents compulsorily behave as they are told.

Hübner et al. (2010) identify two components in organization-oriented programming of MAS:

- “A declarative Organization Modelling Language (OML) – for example, MOISE+ (Hübner, Sichman and Boissier, 2007) and Islander (Esteva, de la Cruz and Sierra, 2002)”, and
- “An organization implementation architecture – for example, S-MOISE+ (Hübner, Sichman and Boissier, 2006) and AMELI (Esteva et al., 2004)” (Hübner et al., 2010: 371).

The OML is used to specify the MAS as organizations of agents in Organization Specifications (OS), and when the agents assume roles in the organization they form an Organization Entity (OE) (Hübner et al., 2010). The final result is an Organizational Infrastructure (OI) that manages operations with respect to organizational objectives, and where the external agents assume roles in it can operate in accordance with rules and policies that make the MAS function effectively as a system. The aim is to make an orderly environment where agents interact towards their individual objectives but at the same time in accordance with the organisation in which they participate.

In Figure 2.1 Hübner et al. (2010) identify a three-layer architecture where can be found the Domain Agents or the External Agents that will assume roles in the OI, the OI itself that will manage and control the MAS through the implementation of services and the

management of the MAS internal states, and the last layer being formed by Agent Platforms, such as Opal (Nowostawski, Purvis and Cranefield, 2002) and JADE (Bellifemine, Poggi and Rimassa, 2000), in charge of the communication among the agents.

The entities in the OI that are responsible for interfacing the External Agents with the organization are the Organizational Proxies. They are in charge of passing all the communication originated from the external agents to the OI and from the OI to the external agents, managing what we call a very controlled conversation where only predicted dialogs can take place. These Organizational Proxies can be found in OI implementations such as *governors* in AMELI (Esteva et al., 2004), *wrappers* in TeamCore (Pynadath and & Tambe, 2003), *OrgBox* in SMoise+ (Hübner, Sichman and Boissier, 2006) and *controllers* in Law-Governed Interaction (Minsky and Ungureanu, 2000).

Our approach differs from those because our agent responsible for the intermediation between the application agents, or external agents, is simply a way for the external agent to use the interaction protocols available for a determined role. These protocols are represented as Coloured Petri Nets that actually define in which conversations the external agent can take part in the institutional environment. These agents might have some degree of autonomy if that is desired by the institutional environment designer. Moreover, these agents make available unmonitored interactions that deliver freedom of speech for the external agents participating in the institutional environment. Freedom of speech makes the institutional environment more like the human organizations where the human autonomous agents have the choice to follow the rules and can engage in negotiations or conversations that are not predicted by the organizations that they are part of.

The OML in our approach is the formalism we have chosen to represent the conversations and the organizational infrastructure of the institutional environment. It is represented as a set of Coloured Petri Nets (CPN) (Jensen, 1992) that defines the agent roles, both for the application agents, or external agents, and the system agents, which are the agents that manage the institutional environment. Therefore the OI is actually the unification of the CPNs of the agents that are present in the institutional environment, and there is no definition of a middleware that forms the OI. In fact, there is no middleware at all. Actually, in our approach we have made available elements that are a way for the external agents to get in the conversation space of the institutional environment, which is completely distributed in the sense that all the administrative tasks are performed by system agents, and

all agents, including the external agents, play their part only in the necessary conversations predicted by the institutional environment designer (see Figure 2.2).

It is important to notice that in our approach the Deputy Agent is part of the OI. It is not merely in the frontier with the OI: it actually implements a CPN that is connected with the system agents CPN forming the OI of the Institutional Environments, differently from other approaches such as Esteva (2003) and Hübner, Sichman and Boissier (2007). The other characteristic that distinguishes the Deputy from the regular proxy agents from Figure 2.1 is that the constraints imposed by the Deputy Agent are less than those of the regular proxy agents. It allows for the use of both predicted and unpredicted conversations through the unmonitored interactions.

The external agents are in the application domain and they actually do not need to be from the same application domain to register with the institutional environment. Furthermore, they can take part in as many institutional environments as they wish or can, as far as they understand the ontologies that define the application domains used by the institutional environments that they register with.

Our architecture is mainly distributed and the Deputy Agents can actually have some degree of autonomy, if the institutional environment designer decides to supply them with some. They do not force the external agents to engage in only predefined conversations; there is the possibility for the external agents to meet in the institutional environment and exercise their autonomy by engaging in unpredicted conversations, as far as they are capable of doing so.

Moreover, our OML is not a customized set of symbols or language. We rely on the strong semantics and syntax of the CPN formalism. All the OI is defined as CPNs that together represent a Normative Space for the Institutional Environment.

Hübner et al. (2010) point out some features of the OI presented in Figure 2.1 that motivate our proposal and their own, which is described in Section 2.5.

In our approach the system agents are accessible by the institutional environment designer when she/he needs to model, for example, particular sanctioning made necessary by the application domain. When we are modelling the institutional environment the designer has the power to define the rewarding/sanctioning policies that will be employed in the

environment. That, of course, depends on the requirements of the application domain that is being worked on.

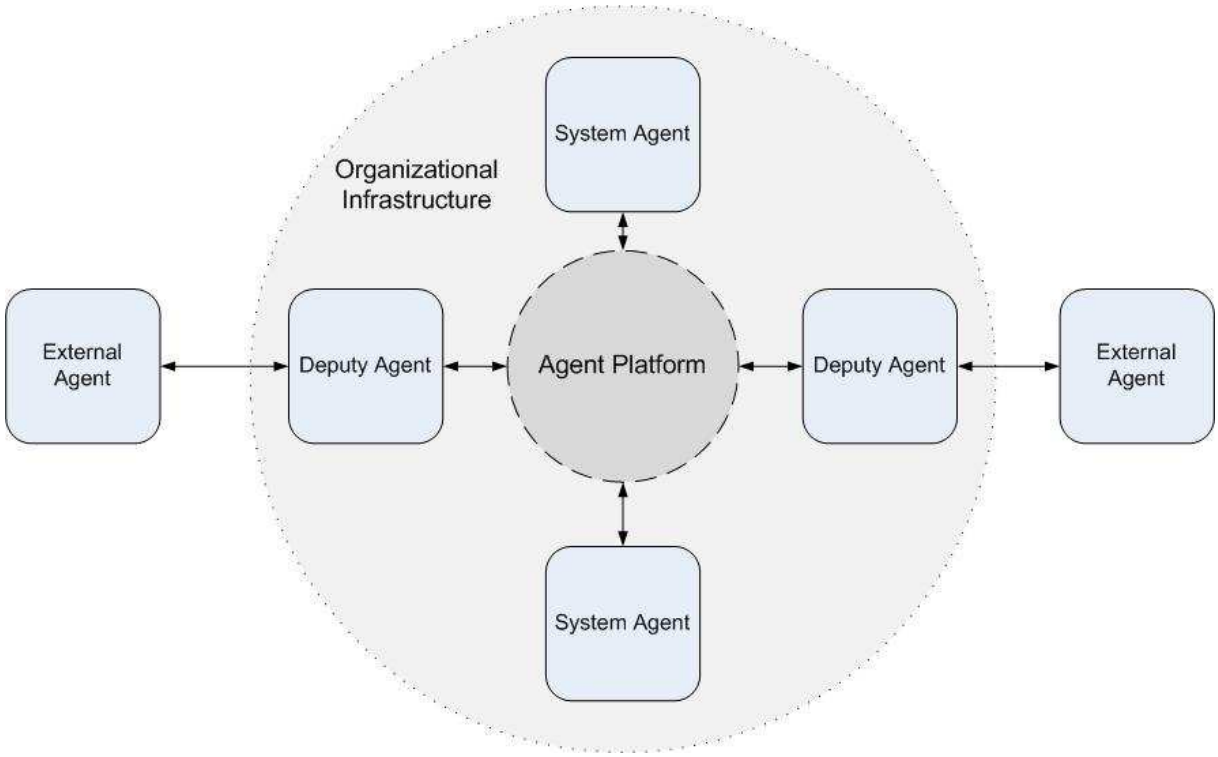


Figure 2.2. Institutional Environment organizational infrastructure architecture

OperA is a framework for modelling agent organizations which uses social order to regulate stakeholders and their interactions (Dignum, 2004). OperettA is a graphical environment that allows the specification and analysis of models based in OperA (Aldewereld and Dignum, 2010). The Alive project (Nieves et al., 2010) uses a Model Driven Development (OMG, n.d.) approach to build systems based on the Service-Oriented Architecture (Bell, 2008). It has three architectural layers: organization, coordination and service. Organizational models based on the OperA framework are built using OperettA to coordinate access to services. These services are invoked dynamically through an organizational context to achieve organizational objectives.

Gaertner et al. (2007) define the notion of a “normative structure” – a formal model for defining how events occurring in one agent interaction protocol (or “scene”) can affect the state of normative state in another scene. They define the semantics of normative structures in terms of a mapping to Coloured Petri Nets. This mapping allows for the definition of a

distributed algorithm to avoid norm conflicts (e.g. obligation vs. permission), in the normative structure by exploiting the CPN concept of state reachability. The normative structure comprises states of a normative scene with transitions fired due to normative positions taken. The main goal is to identify normative structures that are free of conflicts by verifying whether conflicting states are reachable in the CPNs.

We do not only represent norm scenes states in our CPNs. We model agent roles and the norms are modelled as commitments created at design time or by agents at runtime. However, we do not address the analysis of our CPNs to verify desired properties.

Van Riemsdijk, Hindriks and Jonker (2009) explore the way an organization-aware agent can develop organizational reasoning that decides, among other things, (a) when to play roles and (b) if they do play a role, whether they follow norms and how they might respond to possible sanctions. It is argued that agents may develop reasoning about behaving in accordance with the role specification, violating the role specification or adapting the specification of the role to its goals. The agents would have to understand the organizational structure of the MAS and develop behaviour reasoning over that specification and targeting their own goals. The crucial point here is to make the organizational structures understandable by the agents that participate in the MAS.

In our work, even though we use the Deputy Agents to give external agents access to the institutional environment, a norm enforcement policy is used in the sense that agents are motivated to follow the rules but they can break them. This contrasts with other approaches such as OrgBox (Hübner, Sichman and Boissier, 2002) and governors (Esteva, de la Cruz and Sierra, 2002) (Esteva et al., 2004) where regimentation is used and the agents have no options when interacting in the organization unless they follow the norms. Therefore, in our Institutional Environments, organization-aware agents would have more options when performing organizational reasoning to adjust their actions in the organizational environment.

In this thesis we are not concerned specifically about the programming of the external agents to make them organization-aware in our Institutional Environments. However, we define agent roles as CPNs which have well defined semantics and we believe that future work could be done to make the external agents understand the CPNs by themselves, develop reasoning over them, and consequently over the organizational structure of our system.

In Coutinho, Sichman and Boissier (2005) a selection of organizational models are compared in an attempt to create an ontology that describes open MAS as organisations of agents. The work is further developed later by Coutinho, Sichman and Boissier (2009), where more organizational models are added to the study. Initially the models studied comprise AGR (Agent, Group, Role) (Ferber, Gutknecht and Michel, 2004), MOISE+ (Model of Organization for multI-agent SystEms) (Hübner, Sichman and Boissier, 2002), ISLANDER (Esteva, de la Cruz and Sierra, 2002), ODML (Organizational Design Modeling Language) (Horling and Lesser, 2005), AUML (Agent UML) (Parunak and Odell, 2002), and MAS-ML (MAS Modeling Language) (Silva, Choren and Lucena, 2004). Later are added the organizational models TAEMS (Task Analysis, Environment Modelling, and Simulation) (Decker, 1996) (Lesser, 2004), OperA (Organizations per Agents) (Dignum, 2004), STEAM (a Shell for TEAMwork) (Tambe et al., 1999), and AGRE (Agent, Group, Role, Environment) (Ferber, Michel and Baez, 2005).

The study aims to create a meta-model from which unambiguous concepts can be instantiated to define abstract organisational specifications and then generate concrete organizations. They define eight dimensions when making a comparison among the studied organisational models: structural, interactive, functional, normative, environment, evaluation, evolution and ontology. As future work the organisational structure implemented by our Institutional Environments could be analysed in relation to the dimensions listed to see how it fits in the more general and higher level meta model defined by the authors.

2.4. Communication in MAS

Interaction is the basis for those distributed systems, such as MAS, to function well, therefore a great effort in MAS research is dedicated to agent communication. To allow agents to communicate the first step is to define a common language. Agent communication languages (ACL) have been proposed based on speech act theory (Austin, 1962) (Searle, 1969). This theory is based on the observation that utterances must not be considered as simple propositions but as speaker’s attempts that succeed or fail (Austin, 1962).

The definition of ACLs allows for the agents to exchange information and knowledge, and what distinguishes them from other ways of communication are the objects of discourse and their semantic complexity (Labrou, Finin and Peng, 1999). ACLs address

communication at the intentional and social levels (Dignum and Greaves, 2000). Generally, an ACL is composed of three main elements (Genesereth and Ketchpel, 1994): a vocabulary, an inner language to encode the knowledge to be exchanged among agents using the vocabulary offered by an ontology and an outer language to express agents' intentions. The vocabulary should be contained in ontologies shared by the interacting agents. Examples of ACLs used frequently in the implementation of MAS are: FIPA ACL (FIPA, 1997) and KQML (Finin, Labrou and Mayfield, 1997).

More recently, approaches were developed to implement agent communication languages (ACL) based on the use of commitments to manage organizational infrastructures (Colombetti, Fornara and Verdicchio, 2004) (Fornara and Colombetti, 2002) (Fornara et al., 2008) (Verdicchio and Colombetti, 2009). In these approaches an ACL based on the use of commitments between agents to manage the organizational infrastructure is defined. It states what terms should be used to manipulate commitments and even to perform operations on them.

But agents normally do not engage in a single message exchange. Similarly to human beings, they engage in conversations that define the valid sequences of exchanged messages and define the context in which exchanged messages must be interpreted (Labrou, Finin and Peng, 1999). Hence, the notion of conversation as the unit of communication among agents has been promoted (Greaves, Holmback and Bradshaw, 2000). Section 2.4.1 explores interaction protocols as a way to organize conversations among agents.

We share the same view of communicative acts as in Colombetti, Fornara and Verdicchio (2002: 2):

> "We view communicative acts as a kind of institutional actions, that is, actions that are possible on the basis of a set of conventions and regulations, and whose effect is to bring about an institutional effect. More precisely, a communicative act is an action realized as follows: an agent, the sender, sends a message to another agent, the receiver. By doing so, if certain contextual conditions hold, the sender performs a communicative act addressed to the receiver. The effect of such a performance is to bring about a number of institutional facts".

In our approach the institutional facts are established according to the institutional acts that are used in interactions observing the rules or norms of the institutional environment. The institutional acts are defined as the actions identified and necessary for the management of the institutional environment, as well as those of the application domain where the institutional environment is built that might influence the management of the institutional environment. In other words, the ontologies that are used by the institutional environment define the institutional acts used in the institutional environment. These two groups of institutional acts form the vocabulary of the institutional environment and are used in interactions to bring about states of the environment where the institutional facts hold. The institutional facts are used to verify the states of the commitments that external agents assume in our institutional environment.

Later in Fornara and Colombetti (2002) they define commitment-based semantics for agent speech acts. In Colombetti, Fornara and Verdicchio (2004) and Verdicchio and Colombetti (2009) they define the semantics of ACLs in terms of changes in the social relationships between agents, represented in terms of social commitments. In these works they define all the communication among the agents in an institution regarding the use of speech acts that represent actions on commitments, and as well, the interaction protocols when using them.

We do identify institutional acts that deal with commitments because they are the way we enforce norms in our institutional environments. The conversations regarding the use and manipulation of commitments are defined in the roles of the agents that deal with commitments, and actions that the agents execute in the institutional environment generate states where institutional facts are true or false. These are used to verify the state of the commitments regarding their content and condition, explained later in Section 2.7.

2.4.1. Representation of Interaction Protocols

For the specification of conversations among agents the research community has mostly used the Unified Modelling Language (UML), Finite State Machines (FSM), Petri Nets (Reisig, 1985) or extensions of them.

Finite states machines (FMS) are used to model conversations that do not have concurrent paths. Generally, the states in the FMS represent states in the conversation, and the arcs that represent a change from one state to another are labeled by the actions that occur for the

changes of state to take place. Due to its simple way of use FSM are broadly used for conversation representation and modelling, but its lack of representation for concurrent paths has promoted the use of Petri Nets and its extensions. Further relevant characteristics favoring Petri Nets are listed as follows (Cost et al., 2000), (Jensen, 1992) and (Reisig, 1985):

- They have a standard graphical representation;
- They have support for the representation of concurrency;
- They are well studied in the research community and have a standard semantics and syntax for the representation of processes;
- They are applied in real-world applications for the modelling and verification of protocols, workflows and processes;
- There are various tools that support the design and simulation of CPN.

One possible negative side of the specification of protocols using Petri Nets is that they are somewhat more complex then FSMs. On the other hand they offer more power for the representation of protocols, for example, allowing for the specification of concurrent tasks. Complex protocols can, however, generate a combinatorial explosion on the size of the network, which calls for a more careful use of this tool (Cost et al., 2000).

In the work of Baldoni et al. (2008) (2009) a middleware module is developed as an extension to the JADE Framework to provide Organization, Role, and Player classes that are used to implement open MAS as organizations of agents. They implement the powerJADE middleware, and use finite state machines to represent the behaviours of the new classes added as well as the protocols used by players when interacting in the context of the organization. One interesting point in connection with their approach is the possibility of adding new states in the FSMBehaviour that changes the role´s institutional powers at run-time. They do not explicitly treat norms, but their approach does incorporate coordination elements.

In Esteva (2003) concurrency is not allowed within a conversation. They use FSMs to specify agent conversations and the global protocol of the conversation. Properties of the protocols are verified by adapting search graph algorithms and model checking is used to

verify some properties. They advocate for the enforcement of conversations on participating agents as in Nodine and Unruh (Nodine and Unruh, 1999), Martìn, Plaza and Rodrìguez-Aguilar (2000) and Sibertin-Blanc, Hamachi and Cardoso (2000). There is a special Mediator agent, called Governor, that is in charge of guaranteeing that the conversation follows its right and allowed path or pre-specified protocol. The governors are proxy agents that restrict the conversation of the external agents and force them to behave always in a predicted mode.

Coloured Petri Nets (Jensen, 1992) are an extension of Petri Nets that have the same advantages of the regular Petri Nets as well as allowing for the tokens of the net to be of any “colour”. The colour is the type of the tokens that can be stored in the CPN places. Although this adds more complexity to the execution of the network it adds more flexibility in the representation of interaction protocols because the tokens represent data relevant to the conversation that is being modelled. Due to these advantages of Petri Nets on FSM we opted to use Coloured Petri Nets in our approach.

In a CPN the places contain tokens that are of any data type (colour), and connecting places there is a transition which might define operations to be done over the tokens that are selected from the input place. After the operation is executed new tokens can be generated for the output place. The current marking of the net represents the state of the CPN, being it the configuration of the net regarding where the tokens are.

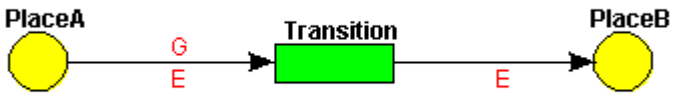


Figure 2.3. Example CPN

Associated with each transition in a CPN there is a guard, which is a Boolean expression that evaluates to true or false triggering the transition. Once the transition is triggered the input arc expression selects tokens from the input place (PlaceA) to be used in the transition action code. The selected tokens are available in a multiset that can be passed on through the output place (PlaceB) or a newly generated set of tokens can be put in the output place. For the guards, arc expressions, and action statements of CPNs there is an inscription language defined to manipulate the tokens and their content.

Specifically in JFern, the CPN tool box that was used in this work, the default arrangement is that there is a transition guard that becomes TRUE if, and only if, all the input arc guards are TRUE. The user can change this and make his own transition guard, if he likes.

AUML (Odell, Parunak and Bauer, 2000) is an extension of UML for MAS design, including the specification of agent interactions. It has an intuitive graphical representation, it is familiar for designers coming from the object-oriented paradigm and there are tools supporting its use. As a negative point, it is not based on a formal model that permits its formal verification, which requires a translation to another formalism for that to be done.

There are works such as the Java-based Agent Framework (Chauhan, 1997) (Galan, 2000) where the interaction protocols are defined in FSM and then translated into Petri Nets. The translation is done for the use of Petri Net tools for the verification of safeness and liveness properties for the interaction protocols specified in FSM.

In Koning, Francois and Demazeau (1998) the specification and verification of agent interaction through the use of Petri Nets is proposed. They propose to model Petri subnets for each agent taking part in a conversation from a state transition diagram. Each agent would be concerned only with the part of the conversation where it is included. The subnets are put together to compose the whole conversation. There is a tool for the simulation, verification and execution of the Petri Nets modelled. Our approach is somewhat similar to this one because we do use connections of smaller nets to form a whole net that represents the normative space of our Institutional Environment. But our smaller nets are connected through FIPA ACL Messages (FIPA, 2002) because they represent agent roles and we use Coloured Petri Nets (CPN) instead of regular Petri Nets. What they call Petri subnets in a conversation space in fact is seen in our approach as the definition of agent roles. These CPNs specify not only the states of a conversation protocol but the institutional state of the agent that has its role represented as a CPN. Is interesting to note that our token can be any Java Object and we do use this flexibility when defining our agent roles, for example, when agents communicate, the tokens are Java Objects that represent FIPA ACL Messages (FIPA, 2000) (FIPA, 2002) and in some parts of the internal representation of the role the token can be other Java Objects.

Coloured Petri Nets were used by Cost et al. (2000) to specify agent interactions. To specify the conversations they used the DesignCPN modelling tool. Their work concentrated on the modelling of conversation protocols and not on the formalization of an

Organizational Infrastructure that manages agent interactions using CPNs, as is main formalism. In our approach we use a Java-based tool named JFern (Nowostawski, 2009) to model and execute our agent roles and consequently the whole Institutional Environment. We do have asynchronous message exchange among the agents in the Institutional Environment, and consequently among the CPNs that represent them. This calls for the management of concurrency and makes CPN suitable to represent agent interactions.

A proposal that agent interactions should be mediated by an entity called a moderator, which guarantees the correct execution of the protocol, was done in Sibertin-Blanc, Hamachi and Cardoso (2000). Cooperative Objects (Sibertin-Blanc, 2001), a high level Petri Net language, is used to specify, validate and implement the Moderators. The moderator actually manages a specific conversation to which it was modelled to, and each agent taking part in that conversation has a thread in the moderator. The moderator will check the communicative acts being used by the agents and verify the state of the conversation. This approach is more like the implementation of a conversation manager that represents its conversations through the use of Petri Nets. A Conversation Manager, such as the one in Purvis et al. (2003), is a piece of software in charge of managing conversations. The conversations can be modelled using Petri Nets, FSM or other formalisms (depending on which formalism the tool supports) and the conversation manager tries to make sure that the right messages arrive in the correct instance of the conversation protocol, which is part of a given conversation in the MAS.

Hanachi and Sibertin-Blanc (2004) extended this work changing the perspective according to which the conversation is seen: Their proposal is to see conversations as processes managed by middle-agents called *moderators*.

In our approach we use CPNs to represent agent roles and the conversations are managed by these CPNs that in fact implement agent behaviours in the Institutional Environment. What leads the conversation are the possible Institutional Acts that are defined for the management of the system and the execution of actions referring to the application domains. The management of the conversations is a low-level task that should be implemented in the Agent Platform that takes care of the exchange of messages among the agents in the Organizational Infrastructure.

In Mazoui, Fallah and Haddad (2002) AUML is used to model interaction protocols that are later transformed in Coloured Petri Nets (CPN) for the evaluation of protocol properties, for

example, if it is free of deadlocks, or if all final states are reachable. The CPNs have one subnet to each role present in the system. This approach is yet again a use of Petri Nets as a validation tool for interaction protocols. We do believe that this is valid and necessary for the correct representation of the concurrency present in the interactions in an Organizational Infrastructure. We use CPNs as the main tool to represent the whole institutional environment defined in our approach from the start to finish of the open MAS's construction.

In our approach we aim to make the conversation more flexible for the external agents showing then, through our Deputy Agents, what the institutional acts are that they can use in the institutional environment and what kind of interactions they can engage with, monitored or unmonitored. In our Deputy Agents it is possible to allow the agents to use among them speech acts not predefined by the application domain designer, which allows for more flexibility of communication among autonomous agents. We believe that Organizational Infrastructures should be prepared to deal with unpredicted conversations, and that in some cases this might be beneficial for the institutional environment to some extent, for example, when we need to gather as many external agents as possible in an organizational environment without putting too many constraints on the behaviour of these agents. This is the case in, for example, Social Networks and Collaborative Systems such as WikiCrimes (Furtado et al., 2010) that our case study explores in Chapters 5 and 6.

It is important to mention that we allow such a flexibility because we believe that true open MAS should be prepared to deal with really heterogeneous and autonomous agents. That is similar to the human model of dealing in their societies; the rules are there, sometimes stricter some times less so. It depends on the application domain and on what are the objectives of the institutional environment designer.

2.5. Social Artefacts

Another approach to implement organizational infrastructures is the use of Social Artefacts (Hübner et al., 2010), where application agents may take part in the management of the organization because they assume institutional roles.

Hübner et al. (2010) sketch a scenario to implement organizational infrastructures. They explain that their approach uses the metaphor of artifacts to implement mechanisms that

facilitate the management of the organizational infrastructure to implement social order (Castelfranchi, 2000) in it. Agents assume roles and use the artifacts to develop actions in the OI. The artifact could be a symbol that imposes authority or a mechanism that facilitates communication, for example. The middleware approach to designing the OI precludes the incorporation of strategies for sanctions and reorganization at the application level.

In our view this approach is only a matter of allowing external agents to play system agents' roles. This is not necessary to implement social order as we have in our model. Moreover, separating the concerns of the system agents from the external agents adds more security and reliability in the operationalisation of the institutional environments.

The reorganization of the environment is implemented in our approach by the redefinition of reward/sanctioning policies, the way the reputation of the external agents is modified and how it is interpreted by the institutional environment, and all this is dependent on the application domain. The rules are simple: the external agents can act in the environment and be monitored when they acquire commitments in the environment, being punished or rewarded depending on the outcomes of their actions regarding these commitments. Changes in the interpretation of the rules will depend on the implementation of the external agents, their degree of autonomy and how they evolve in time – for example, how they adapt to the rules of the game. By that we mean the norms of the system.

The change of roles in the institutional environment can also be seen as a reorganization of the organizational infrastructure. This means that when an external agent changes roles, by being promoted to it or by its own choice, it will have a new portion of possible predicted dialogs attached to its CPN that will reorganize the institutional environment as a whole, because it is composed by the merging of all the CPNs that represent external agents and system agents.

Artefacts play the same role as the proxy agents in the regular organizational infrastructures. They are not autonomous and actually are an interface between the external agents and the environment.

While we use CPNs through Deputy Agents to allow external agents to perform institutional acts in the Institutional Environment, Tinnemeier, Dastani and Meyer (2009) have developed a programming language to implement norm-based artifacts that regulate

external agents when performing actions in an agent organisation. The artifacts provide the interface between external agents and an organisational platform.

2.6. Commitments between Agents for Social Control

Commitment between agents are agreements defined between two parties, a debtor and a creditor, stating certain propositions that are evaluated to change its state (Colombetti, Fornara and Verdicchio, 2004) (Singh, 1999). The commitments are evaluated and might influence the status of the agents in an MAS. The management of the commitments in an MAS can be used for the definition of reputation of agents in the MAS because their states tell if the agents are doing what they have committed to do in the institutional environment.

An example of the use of commitments in a MAS would be if a buyer agent commits to pay for a good and a seller agent commits to deliver the good. The actions of those agents in the MAS will cause the change of state of these commitments and they can be tagged as good or badly reputed agents in the MAS.

Commitments are generally treated as objects that have states and a life cycle (Colombetti, Fornara and Verdicchio, 2004) (Singh, 1999). Operations to change the states of the commitments are defined and they are used according to the evaluation of actions of the agents that are debtors or creditors of the commitments. They have also a condition to become *active* and a content that is evaluated to check if the commitment has reached on of their final states.

The condition and content of the commitments have a temporal component. They need to be checked against facts that occur in a time period. For example, when certain conditions hold on a certain time the commitment becomes active and if the seller delivers the good in a maximum of 10 days after the order date it has kept its commitment to deliver in 10 days the good. Here the necessity is identified of implementing a way to check the condition and content of commitment against a history of facts that happen during a certain period of time.

Commitments are used in our approach as a way to implement norms in the open MAS. They have been used before to serve as a mechanism to implement social control in organizational infrastructures (Singh, 1999) (Colombetti, Fornara and Verdicchio, 2002).

We use commitments to implement social order in our institutional environments. A specific system agent is responsible for doing operations on them and that is modelled in the interaction protocols that are part of the CPN that defines the role of the system agent and an external agent that joins the open MAS. The state of the commitment objects and their verification is managed by the institutional environment through systems components. These components are responsible for managing the commitment objects in the institutional environment. The social order implementation has the commitment objects verification outcomes as inputs, and the implementation of reputation and trust mechanisms help in the organization of the institutional environment.

Artikis, Sergot and Pitt (2009) provide a theoretical and executable framework for the specification and execution of open MAS using two action languages: C+ and Event Calculus. First they create specifications in C+ of the social constraints for specific open MAS as a transition system composed by social states, and then use that as input to the CCalc tool that will run queries on the specification. The constraints can be, for example, permissions, obligations and sanctions associated with members of the open MAS.

There are three types of queries: prediction queries which starting from an initial state, observing a narrative of facts and social constraints, offers as a result a social state; planning queries which allow for the agents to compute plans to avoid run-time conflict at design time or to update plans at run-time; and postdiction queries that compute past states of the open MAS.

The SV tool developed in Prolog executes prediction queries over an Event Calculus specification given a sequence of temporal events to predict the social states of the Contract Net Protocol at run-time. After events occur, prediction queries compute the current state of the open MAS regarding, for example, permissions, obligations and sanction.

In the context of our work, norms are specified via the definition of commitments at design-time and at run-time as well. These are defined in the CPNs that specify the roles of the agents in the open MAS. Execution runs over those specifications could be simulated at design time to examine and evaluate situations that can lead to undesired outcomes at runtime (but this possibility is not investigated in this thesis).

A formal normative framework can be defined in the context of agent-based systems, such as in the work of Lopez y Lopez, Luck and d'Inverno (2006). We do not specify norms in

an explicitly formal representation. Instead, in our work we use commitments to provide the social guidance among agents in institutional environments.

2.7. Monitoring and Verification of Commitments

The commitments represent the norms in our model. Below, we describe a couple of recent approaches in the verification of norms in MAS and translate their suitability to our requirements. It is important to keep in mind that to verify a commitment, in our approach, we need to check the temporal propositions that represent their condition and content against some representation of states of the institutional environment describing facts that happen in it.

To coordinate and control agents in a MAS context, Astefanoaei et al. (2009) identify three approaches:

- coordination artefacts (Arbab, 2004) (Gelernter and Zuck, 1997) (Ricci, Viroli and Omicini, 2007),
- the application of social and organizational concepts through management mechanisms (Ferber, Gutknecht and Michel, 2004) (Dignum, 2004), and
- the combination of the last two in what they call organization-based coordination artefacts (Dastani et al., 2008) (Tinnemeier, Dastani and Meyer, 2008) (Boella and van der Torre, 2008) (Esteva et al., 2004) (Hübner, Sichman and Boissier, 2002) (Hübner, Sichman and Boissier, 2002).

Astefanoaei et al. (2009) introduce a programming language that is designed to facilitate the implementation of norm-based organization artefacts. The artifacts are the way the agents sense and interact in the environment, they use norms to identify violations and sanctions to punish the agents that have violated the norms.

The external agents produce facts that hold in the organizational environment after they execute actions in it. The facts can be seen as pre- and post-conditions of actions that are executed by those external agents (Astefanoaei et al., 2009). Therefore, what exists in reality is a set of facts that represents part of the state of the organizational environment, and the external agents' actions change the facts and consequently the states.

Astefanoaei et al. (2009) consider norms as Counts-as Rules (Searle, 1995). They make a distinction between institutional facts or facts that have an institutional effect or meaning, and brute facts that are more concerned with the mechanics of the organization. We make the same distinction in our approach when using counts-as rules to separate these concerns in our Institutional Environments, discussed more in Section 2.8.

In our approach it was necessary to define a way to represent the content and condition of the commitments using a temporal logic because of the intrinsic characteristics of these commitment attributes, as explained in the last section. Moreover, the temporal propositions needed to be verified against a linear history of facts that define states in the institutional environment. Programming the whole institutional environment as norms does not suit our approach because we have the norms distributed throughout the environment in the form of commitments. The commitments are set only when necessary due to some conversations that need to be monitored or have agreements formalized. These conversations are identified as the ones that influence the institutional environment on the reaching of its general goals, which is the good functioning of the institution according to its rules.

In Alvarez-Napagao et al. (2010) a production system is used in the form of a forward-chaining rule engine to implement the triggering of rules that check past events against norms in a MAS. Their intention is to allow for the addition of rules in the production system at runtime, and with this make possible the representation of rules unpredicted at design time and possibly only necessary after a certain scenario evolves in the organization.

This last approach suggests the use of a knowledge base constantly updated with facts that will serve to verify the norms of MAS. This is particularly interesting for us because we believe we need this knowledge base as well. Furthermore, it needs to be developed in a way that our commitments can be checked against it.

Cranefield and Winikoff (2011) focus on modelling the general concept of social expectation and demonstrate the use of model checking for detecting the fulfilment or violation of such expectations by extending the MCLITE and MCFULL algorithms of the Hybrid Logics Model Checker (Dragone, 2005). They state that the advantages of building on model checking, rather than implementing a new checking algorithm (as was done previously (Cranefield, 2006)) are that they are “working within a clearly defined and well studied verification framework, and that it allows existing software to be extended

including a range of optimisations that have been developed for model checking" (Cranefield and Winikoff, 2011) on page 2. Although the problem of model checking, in its full generality, is very complex, the problem of model checking a path (a finite or ultimately periodic sequence of states) has also been studied and "can usually be solved efficiently, and profit from specialized algorithms" (Markey and Schnoebelen, 2003) on page 251. They have therefore investigated the applicability of model checking as a way of checking for expectations, fulfilments and violations over a model that is a linear history of observed states.

The theory underlying the Cranefield and Winikoff approach is designed to apply equally well to both online and offline monitoring of expectations, a distinction that has not been made in previous work. This distinction is described as follows:

> "For online monitoring, each state is added to the end of the history as it occurs, and the monitoring algorithm works incrementally. The underlying formalism can assume that expectations are always considered at the last state in the history. In contrast, in the offline mode, expectations in previous states are also checked. At each past state, the then-active expectations must be checked for fulfilment without recourse to information from later states: the truth of a future-oriented temporal proposition φ at state s over the full history does not imply the fulfilment at s of an expectation with content φ" (Cranefield and Winikoff, 2011: 2).

In our approach we needed a way to check the content and condition of commitments between agents against a history of facts that would constitute states assumed by the institutional environment in time. Therefore, we have used the expectation manager developed by Cranefield and Winikoff, to monitor the condition and content of commitments defined in the institutional environment. The expectation manager implements a model checker that makes it suitable for the checking of the temporal propositions. These propostions represent content and condition of commitments against a linear history of facts that record the occurrence of actions performed by agents in the institutional environment.

Our approach uses the online monitoring capability of the expectation manager to check a model constantly updated. This update occurs by the introduction of new facts that represent actions that are happening in the institutional environment. These new facts create

new states that are added to the history managed by the model checker (See more details in Section 4.5).

2.8. Counts-as Rules

The designers of organizational infrastructures confront a problem when representing the information or facts that represent the states of the organizational environment: The mixing of concerns with the domain or context of the organization and the low-level facts that are more related with the functioning or execution of actions in the organizational setting (Aldewereld et al., 2010) (Grossi et al., 2006). It is not uncommon to mix these layers of abstraction for information representation, and the counts-as rules appear as a mechanism that makes it possible to represent the information from different levels of abstraction and at the same time connect the layers in rules that are fired after low-level facts appear in the organizational environment (Paschke, Dietrich and Kuhla, 2005), (Aldewereld, 2007) and (Dastani, 2009).

In our approach we represent commitment content and condition through temporal propositions that are checked against a model that represents the state of the institutional environment regarding the facts that occur in it due to actions of the registered agents. When defining the commitments we got to the point where concepts and facts related with the application domain were getting mixed with concepts and facts from the institutional environment internal functioning. This was causing a mix of concepts from different levels of abstraction in the definition of the commitments between agents. We opted for the use of counts-as rules for the better accommodation of the low-level facts related to the institutional functioning in the commitments that must take them into consideration to evaluate more abstract facts related with the application domain.

The mechanism used to implement this approach was the gathering of low-level facts that would translate to an institutional effect necessary for the evaluation of a condition or a content of a commitment against the model defined in the model checker. An example is the necessity of some system agents being online in the checking of a condition that would change the state of a commitment to *active*.

2.9. Achieving Social Control in Normative Systems

Our model is built along the lines of a social control approach that consists of developing an adaptive and auto-organized control set up by the agents themselves. As it is intrinsically decentralized and non-intrusive to the agents' internal functioning, it is more adapted to open MAS than other approaches, like centralized institutions (Arcos et al., 2005).

In MAS the decentralization and autonomy of their components, together with the association of openness to it, bring flexibility and adaptability to their use. The resources and expertise are distributed among the agents that participate in the MAS, and through openness autonomous and heterogeneous agents built by third parties can become part of the system without explicit concerns on how the others were implemented. Therefore, these characteristics have to be taken into consideration when managing interactions in MAS (Muller and Vercouter, 2010).

Social control distributes the management of the system among the agents that are part of it hence it being adapted to the MAS domain. The agents have to cooperate to perform the global control of the system. Trust and Reputation models are used to implement social control in MAS – these concepts are applied in a way that agents build reputation in the system and trust on other agents based on their past interactions. The consequence of this approach is the social exclusion of agents that develop an undesired behaviour in the MAS.

Our approach for social control aims to implement the social exclusion of badly behaving agents but it does not add complexity in the calculus of the reputation of an agent. We limit it to the observation of the fulfilment of commitments assumed by the agent with other agents that play external roles in the institutional environment or commitments assumed with the system agents that manage the institutional environment. The latter are defined as the social commitments that the external agent assumes when it registers with the institutional environment and acquires the right to play a role in the open MAS. We avoid the inclusion of agent recommendation or gossiping in the calculus of the reputation of an agent, and although this is not forbidden and the agents can use their autonomy and freedom of speech to give recommendations to other agents in the institutional environment, it is up to the internal inference implemented by the agent in the building of trust in other agents to take into consideration the referring or the gossiping.

The addition of referring and gossiping in the calculation of the reputation score might be necessary in some cases according to the application domain. But, we leave the decision to use it for the system designer because our purpose is to allow for the designers to have it if they need a more complex way to calculate the reputation score for an external agent. In our approach we manage commitments and evaluate their states to change the reputation score of the external agents. That might be seen as a limitation for some designers and they can build up more complex formulas for the calculation of the reputation score according to the explored application domain. We see our approach as a sufficient and simple way to implement social order in the institutional environment; future work can be done in this area to experiment with more sophisticated ways to calculate the reputation score, which, for the moment, is out of the scope of this thesis.

Furthermore, the reasoning about the reputation that the agents acquire in the institutional environment used to build trust in other agents is left to the agent itself. Therefore, the agents have all the autonomy to decide with whom they will interact; the reputation built in the institutional environment is implemented so that the agents are aware of the bad or good behaviour of each other, and that should lead to social exclusion if the agent is badly behaving, but the ultimate choice is left to the agent itself. The sanctioning is implemented according to the necessity of the application domain and the institutional environment designer. They have the choice to let the system be more constrained by forcing the agents to assume commitments in the system through monitored interactions. This is demonstrated in later chapters of this thesis. We aim to allow for the adoption of strategies that have as goals the inclusion of badly behaving agents in interactions that attempt to develop some transformation in their behaviour or sell some idea that should be analyzed by the badly behaving agents.

Castelfranchi (2000) identifies the social order of a MAS as the state where the system is actively functioning pursuing the achievement of a goal or completion of a global task, even though it is comprised by autonomous and heterogeneous agents with self-interests. Therefore, social order is applicable in the context of open MAS.

Institution approaches (Minsky and Ungureanu, 2000), (Arcos et al., 2005) and (Hübner, Sichman and Boissier, 2007) are suitable for the use of social order because they define rules that should be followed by the agents and sanctions to be applied if they are broken.

The concern here is to try to avoid building a central entity that has a global view of the system making it difficult to apply this approach to open systems.

In our approach the agents are free to exercise their autonomy; this is actually a great aim of our research efforts. External agents have an assistant agent that will present the way it can interact with our institutional environment in the form of a Coloured Petri Net (CPN). We took care in elaborating a CPN that gives freedom for the agents to follow regulations interacting in pre-identified and defined interaction protocols, but our approach also allows for interaction without them. The agents have autonomy to use their internal reasoning to engage in unmonitored interactions with other external agents that are members of the institutional environment. They also have the choice to choose with whom they will engage in interactions based on the reputation that other agents build in the institutional environment, but they can put their own history of interactions in the decision of who to trust. The mechanisms implemented in our approach are more in line with the social control approach.

Adaptation and auto-organized control are characteristics of the social control approach, and trust and reputation models (Castelfranchi and Falcone, 1998) (McKnight and Chervany, 2001) (Sabater, 2002) (Conte and Paolucci, 2002), (Melaye and Demazeau, 2005) are often used to implement it.

In our approach the reputation is built based on the positive or negative outcome of the commitments assumed by the agents in the institutional environment. The information about the reputations of the agents that have registered at some point in time with the environment are available and the decision of trusting an agent is left to the reasoning of the other agents prior to engaging in conversations with it, based on its reputation.

Social order is reached because the interactions that should compulsorily be monitored by the specification of commitments are modelled as such, and their outcome will influence the reputation of the agents in the institutional environment. However, there is always the possibility of an agent engaging in conversations without checking the reputation beforehand; we do like that agents are allowed to exercise their autonomy. It might be necessary to engage in conversations with agents that have bad reputations, or the reputation they have may not be important.

We prefer to rely on authorities of the institutional environment for the calculation of the reputation that relies intrinsically in the outcome of the evaluation of the external agents' actions in relation to the commitments they assume in the institutional environment. Furthermore, the neighborhood referred by Castelfranchi (2000) is implemented in our approach as the agents that have interacted at least once with the agent that is building trust in other agents.

2.10. Norm Enforcement and Regimentation

In an organizational specification the norms represent the way the designer will represent the goals of the organization as a system. The norms will guide or cause the agents to develop certain acceptable behaviour in the organization in the sense that as a system it has objectives and conditions to achieve success on its purpose.

In the context of our work, norms have a similar meaning but are represented as commitments between agents. The commitments can have a particular meaning when defined between external agents or a social meaning when defined between external agents and system agents. The first is the agents that join the institutional environment and the last the agents that manage it. We employ a similar representation for our commitments as in Colombetti, Fornara and Verdicchio (2004). Their commitments have a timeout to become *active* and a deadline to have their content *fulfilled*, and we employ the same policy.

As stressed by Grossi, Aldewered and Dignum (2007), norms without enforcement mechanisms are worthless. The approach of Hübner et al. (2010) considers two types of mechanisms to make the norms be observed in an organization:

- Regimentation for preventing the agents from breaking norms, or making them behave as expected, and
- Enforcement to be used after the norms are broken and some measures are taken to persuade the agents to not break them again.

2.11. Summary

In this chapter, related work was presented and, when pertinent, comments on details of our approach to the modelling and implementation of open MAS were made. The next chapter

specifies all the mechanisms and methods used to build open MAS as Institutional Environments.

Chapter 3

Institutional Environments

In this chapter the concept of an Institutional Environments is defined and all its elements are specified. The conceptual framework is defined from Section 3.1 to Section 3.6. Details on how the elements introduced in these sections are implemented as CPNs are given in Section 3.7. The high level architecture of the Institutional Environment is shown in Section 3.8, and the reusable elements to be used when making an Institutional Environment are highlighted in Section 3.12. A better view of what customizations are necessary to make an Institutional Environment can be found in Chapters 5 and 6, where there is the definition of the WikiCrimes domain and execution of scenarios in that domain respectively.

3.1. The idea behind Institutional Environments

In this chapter we present our approach to modelling Open Multiagent Systems as Institutional Environments. Therefore, we elaborate here on the elements that form the base structure for the construction of institutional environments. Human institutions (North, 1990) are organized in a way that the heterogeneous agents, here the humans, interoperate and collaborate in tasks observing rules or norms that dictate the way they should behave to comply with regulations that make the environment, where they are found, organized and driven to common goals. We see these as heterogeneous groups that form institutions to organize and manage human groups, such as universities, companies and even soccer teams. Our approach to build institutions is to set an artificial environment where heterogeneous agents, humans and software can join and interoperate in it, observing the

rules of the environment. In an institutional environments there are ways to guarantee that it is in charge of enforcing their rules and punishing those agents that violate them or rewarding those that follow them.

The institutional environment is built upon an architecture that gives more autonomy to the agents in the sense that there are no strict interface agents involved in the process of interoperation as in Esteva´s approach (Esteva, 2003), but instead we have a normative workflow that guides the agents in their interactions in the environment. The normative space has implemented in it mechanisms to contemplate interactions unpredicted at design time, allowing the free behaviour of the agents in the institutions as in their human counterparts. The reputation of the agents is built upon their behaviour, and it associates with the agents a label that can be verified before others start interacting with them implementing social control.

To deal with the autonomy of the agents in our institutional environments, the normative workflow implements through its components a mechanism to label the agent with a reputation score in the institution. This reputation represents a measure of goodness of the agents and shows how they behave in the institutional environment, allowing for the implementation of reward and punishment mechanisms to control the cooperation of the agents and to induce the agents to behave according to the rules of the institutional environment, as in human institutions.

The idea is that the agents join the institutional environments and make use of a high degree of autonomy while interacting in these institutions. As in human institutions the rules can be broken and the agent can cooperate or not with the institutions by following the rules or not. The reputation mechanisms are there to motivate the agents to behave according to the rules. Agents are likely to cease interoperations with an agent that has a bad reputation, and it will be without partners to engage in interactions, leading to its social exclusion in the institutional environment. The outcome of that kind of scenario would be that the badly behaving agent would have no option other than to leave the institution or to follow its rules to get a better reputation and recover its credibility in the institution.

The kind of system we are targeting involves the management of communication among heterogeneous agents developed by different people and therefore a common language and ontology must be defined to standardize the message exchanges. The conversations in the institutional environment will happen in accordance with the activities performed by the

agents; what is a valid dialog is defined by the interaction protocols that together build a context where the conversations take part. The protocols must be followed to deliver the outcomes of dialogs predicted at design time, and for an unpredicted dialog there is a generic protocol that must be followed to exchange messages. The unpredicted dialogs are treated as the Freedom of Speech feature (see Section 3.10). It generalizes a generic dialog that heterogeneous agents can use to interact. An agent can, for example, decide to create commitments that were not defined at design time or state anything it wants to its partner in this kind of dialog without violating the protocol.

During conversations the agents can perform actions that will generate commitments with others to do an activity or achieve a state of affairs, for example, the commitment to pay for a commodity after a conversation where one agent buys it from another agent, or the commitment to deliver that commodity after the first commitment has been *fulfilled*, meaning that the payment has occurred.

Norms are represented in institutional environments as commitments acquired by the agents when they go through some conversations. These commitments are managed by institutional environments through system agents, and the life cycle of these commitments will determine the computation of the agents' reputation in the institutional environment. That view is not a standard view of norms where, for example, prohibitions are set as well. We have a social view, where the implementation of norms as commitments generate expected patterns of behaviours by the agents that integrate the institutional environment. The agents can start not behaving as expected but, after being punished and losing reputation points, they can decide to change behaviour and comply with the social order of the institutional environment being more accepted in it.

A commitment is an object with a life cycle (see Figure 3.2), and states that change with time and according to the actions the agents carry out. They are stored for consultation if required by the parties involved. The commitments are seen in the institutional environments as a kind of contract between two agents in relation to an action that must be performed by one of them so that some norm from the institution be followed. An example of that would be a commitment between two agents where one commits to pay for some commodity to another if it wins the auction. In the Auction institution this commitment expresses the institutional norm of an agent having to make the institutional action of paying for the commodity if the institutional fact of winning the auction becomes true.

Further details on the life cycle of commitments are explained later on in this chapter (see Figure 3.2).

Agent roles represent the behaviours of an agent related to the possible conversations it can take part in the institutional environment. The agent roles, played by external agents, are represented by Coloured Petri Nets (CPN), and they, together with the CPNs from the system agents, form the normative space of the institutional environment. The normative space can be seen as a CPN of the whole system that could be analyzed in relation to all the formal aspects of CPNs[1].

The normative space of the institutional environment is identified as the workflow of the whole institution. In this workflow we can see in general terms how the whole institution functions in terms of how the roles interact to realize the activities in the institution. Figure 3.8 represents the top-level architecture of our framework and the interoperation among the various components shown in that figure is described in Sections 3.7 and 3.8.

The workflow of the institution is not maintained by any middleware implemented; this is noticed when modelling the institutional environment by the relationships among the system agents and the agents that play external roles in the system. The activities that comprise the institution put together certain agents in certain moments and it is in these moments that the workflow of the whole institution is noticed. The institution is managed through the development of the agents' actions and the observation of the norms of the system implemented through the commitments and the observation of the agents' reputations that are part of the institution. The flow of messages among dialogs or performance of activities represents the workflow of the institutional environment. Therefore, the workflow is distributed in the agents that compose the institutional environment and they can be spread across different locations, since the connection among those agents happens through FIPA ACL message passing.

The workflow specification is a tool for the designer of the institution. In the workflow it is possible to see and verify where the agents take part in the institution according to their roles. How the activities are developed is strictly modelled at design time because there is no management of it by any middleware; the analysis can be made regarding the characteristics of the CPN in a CPN simulator. The normative space that forms the

[1] This kind of analysis is out of the scope of this thesis.

institutional environment is formed by the composition of the CPNs from system agents that organize and manage the environment, and the external agents that register with the institutional environment.

The workflow interconnects subnets that represent the agents involved in the conversations in the institution. The set of subnets represents all the roles comprising the institutional environment, and the relationships between them represent all the conversations that can be carried out in the system.

Connecting the CPNs is the message-passing layer offered by the Opal platform (Nowostawski, Purvis and Cranefield, 2002). Opal offers all the mechanisms necessary for asynchronous FIPA ACL Message exchange. And, using the JFern Tool Box (Nowostawski, 2009), we model all the CPNs that can be identified in the institutional environment: the system roles, the external roles and the higher level CPNs that connect these subnets forming the normative space and representing the workflow of the institution.

3.2. Institutional Acts

In an institutional environment a group of actors agree to follow a set of regulations to develop a fruitful relationship with the others participating in it. That set of regulations is formed based on the actions each individual can perform in the institutional environment, according to the role they play. In other words, institutional actions (Colombetti, Fornara and Verdicchio, 2002) taken by agents in an agent society must follow the rules imposed by a certain set of regulations defined by an institution. Those institutional actions are in fact the speech acts identified in the context of an institution that are used as illocutions in conversations and are used here to predefine courses of actions and build interaction protocols.

We distinguish and associate institutional acts and brute actions in the following way: the first defines actions that will generate institutional facts, that is, facts that are relevant in the context of the institution, for example, when two people sign a wedding contract (institutional action) the institutional fact “These two people are married” is validated as true in the institutional environment. Brute actions can generate brute facts and institutional facts, for example, raising your hand in an auction represents the brute fact of signalling or calling attention, but in a contextualized institutional environment, such as an auction

house, it can count as an institutional action of bidding, which will generate an institutional fact “Someone made a bid”. In our approach we use the notion of counts-as to separate brute acts needed to manage the institutional environment from the institutional acts that are more related to the application domain where the institutional environment is built. For example, the brute facts that some system agents are online and that one of them has received some specific type of message count-as the institutional fact that an important report has its status accepted.

The institutional environment represents well-defined groups of agents that together form organizations that follow a set of regulations specifying how agents should undertake activities in a specific domain. Therefore, we use institutional actions to identify standard dialogs that take place in an institutional environment and define CPNs that will manage the interaction protocols necessary to achieve the wanted outcomes. Some examples of institutional actions would be to bid in an electronic auction institution, buy in an electronic commerce institution or to kick in an electronic soccer game institution. The process of identification of the institutional acts in a domain requires analysis and evaluation of all the elements defined in that context. For that we specify ontologies that define the concepts identified in that specific domain and relationships among them. Further work could be done so that these ontologies can be extracted from ontology servers as in Cranefield and Purvis (2002) and Cranefield et al. (2002), or from representations that are found in the semantic web (Lee, Hendler and Lassila, 2001). Since these ontologies will contextualize the application domain for the agents involved in the institutional environment, it would be necessary that they make a distinction between brute actions and institutional actions. Or in other words, the ontology would be contextualized in relation to the institutional environment that is implemented in that application domain[2].

In the work of Colombetti (Colombetti, Fornara and Verdicchio, 2002), some communicative acts are seen as a kind of institutional action, i.e., actions that conform to a set of conventions and regulations, and whose effect is to bring about an institutional effect (Colombetti, Fornara and Verdicchio, 2002), (Verdicchio and Colombetti, 2009). In other words, institutional actions taken by agents in an agent society must follow the rules imposed by a certain set of regulations defined by an institution.

[2] Further study on the specification and implementation of extraction of ontologies of the kind explained here from ontology servers or the semantic web is out of the scope of this thesis.

In Figure 3.1 we can see an example of an institutional structure where two institutional environments that model different Open Multiagent Systems are found; those institutional environments represent well-defined organizations that obey a set of regulations, which specify how agents should undertake activities in a specific application domain.

Institutional Acts help in the identification of dialogs that take place in the Institutional Environment. The identification of the institutional acts relies on the ontologies that define the application domain. Once identified, the institutional acts will form a kind of Context or Content Language for that specific institutional environment, because the terms identified will initiate or be part of interaction protocols that form the workflows of the agents registered with the institutional environment.

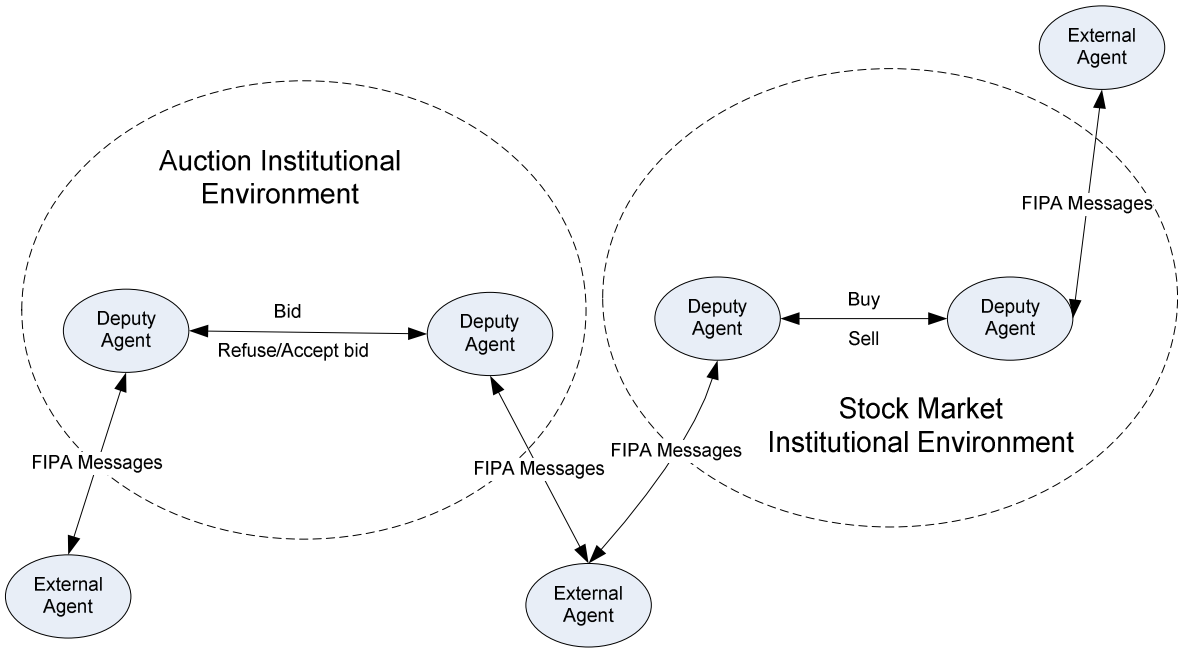


Figure 3.1: Institutional Environments and Institutional Acts

The Agent Communication Language (ACL) is the mechanism by which agents perform communicative acts. They are intended to be used generically and independent of context, which makes their ongoing development difficult. By the use of institutions and with the identification of institutional acts we expect that the contextualization of ACLs to specific domains will be made easier.

The process of identifying the institutional acts in a domain, Figure 3.1, demands analysis and evaluation of all the elements defined in that context. For that we examine ontologies that define the concepts and relationships among them identified in that specific domain.

In the process of identifying the institutional acts of the institutional environment there was the identification of two separate groups:

Institutional Acts from the System: these represent acts that are used when doing interactions that relate to the management of the institutional environment. Therefore, they are considered generic and necessary to any institutional environment built using our approach, e.g. make commitment, set commitment state to be *fulfilled*[3];

Institutional Acts from the Application Domain: these are acts that are used specifically in the context where an application domain is defined. They generate the necessity of redefinition to every new institutional environment that is defined using our approach, for example, biding, buying and getting married.

3.3. Coloured Petri Nets to Model Conversations

Petri Nets have long been used for the modelling and implementation of interaction protocols, workflows and concurrent processes in general (Jensen, 1992) (Cost et al., 2000) (Gutnik and Kaminka, 2006). Existing tools for Petri Nets are used to verify safety and liveness properties for the specifications generated. Concretely, there are software environments which permit the specification of protocols and that execute the Petri Net in a simulator to verify the protocol.

We use Coloured Petri Nets (CPN) to model the normative space of the institutional environment. Each agent in the environment has a role in the system. This role is modelled as the workflow of the agent in the institutional environment. The workflow of the agent has all the possible institutional acts that the agent can use in the institutional environment, so it models all the possible conversations that the agent can take part in the institutional environment.

We call the model of the role of the agent a CPN of the agent's "workflow" because it is going to be part of a bigger CPN that is composed by all the "mini" workflows that represent agent roles. These "mini" workflows will form what we call the normative space of the institutional environment, which in fact is the workflow of the institutional environment as a whole (Figures 3.3, 3.4, 3.5 and 3.7 presented later in this chapter).

[3] These kind of Institutional Act can be used to develop an extended ACL suited to institutional environments.

In our approach it is possible that some dialog exists among agents that are not anticipated by the institutional environment model. The institutional action for that kind of dialog is “unmonitored interaction”. That is allowed to add more flexibility to the agents that want to interact without assuming formally any compromises with other agents and, of course, it is a job for the external agent itself to understand and predict this kind of interaction. An autonomous agent present in the institutional environment will have the power to engage other agents using unmonitored interactions and exercise its autonomy engaging in conversations not predicted completely by the institutional environment.

3.4. Commitments for the Implementation of Institutional Norms

Commitments are objects that are managed by a special System Agent, called a Monitor Agent. This agent is responsible for monitoring interactions where commitment objects are used to formalize agreements between agents. The Monitor Agent stores the commitment objects and executes operations that will change their state (that will be covered in more detail in Section 3.7.2).

Commitments objects have a life cycle, extracted from Colombetti, Fornara and Verdicchio (2002), represented in the Figure 3.2 below, and obey certain rules to change their state through the use of some operations on them. Basically, the change of state of the commitments will depend on the evaluation, by a model checker, of temporal propositions that define the condition and content of commitments, as introduced in Section 2.6. The model checker is implemented in a System Agent called the Institution Agent and the execution of operations that create and change the state of a commitment is managed by the Institution Agent and the Monitor Agent together. Details of these agents will be introduced later in this chapter.

In Figure 3.2 we can visualize the states that a commitment can assume in the institutional environment. The change of states occurs through the use of operations on the commitment and the evaluation of temporal propositions that represent the condition and content of a commitment. The Monitor Agent uses the operation MakeCommitment to instantiate a commitment object and the operation SetCommitment to change its state.

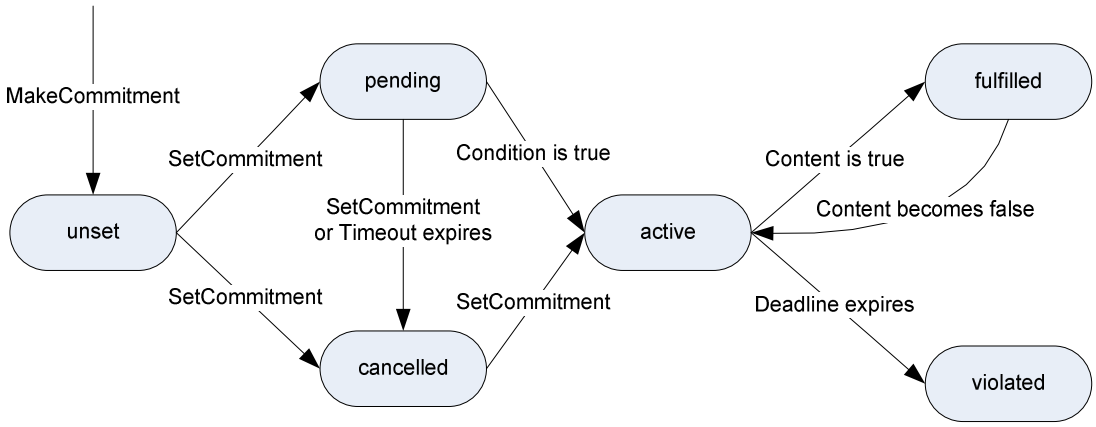


Figure 3.2: State machine of the lifecycle of commitments adapted from Colombetti, Fornara and Verdicchio (2002)

A commitment object has the following attributes:

- Identifier: to uniquely identify the commitment.
- Creditor: the agent that owns the commitment.
- Debtor: the agent that is committed with something for the creditor.
- State: the actual state of the commitment.
- Condition: a formula expressed in temporal logic to be evaluated against the model (a history of observed states, each represented as a set of propositions describing facts that occurred in that state), that is present in the model checker implemented in the Institution Agent. If the formula is evaluated to true in a certain period of time it can change the state of the commitment to *active*.
- Content: a formula expressed in temporal logic that will be evaluated by a model checker present in the Institution Agent. The model checker evaluates whether the actions taken by the debtor have reached a point that satisfies the commitment, changing its state to *fulfilled*, or whether the agent could not reach the state desired by the creditor. A deadline may be associated with the commitment to indicate that the debtor has a time to reach the *fulfilled* state. In the case of this deadline not being reached the state of the commitment will be set to *violated*. If a commitment with content evaluated to true is then evaluated to false it become active again, which is a different approach from the one found in Colombetti, Fornara and

Verdicchio (2002). This change of state is necessary when there are commitments that initially are considered fulfilled and should be like that all the time. If in a time in the future the content becomes false then it returns to active because it is still valid and needs to hold in the institutional environment.

When a commitment object is in the *unset* state, it is called a 'pre-commitment'. In that situation no agent is in the state of being the accepted debtor of the commitment yet. Basically, a commitment is created when a monitored interaction is required in the workflow of an external agent or a system agent. The creditor asks the monitor agent to monitor its interaction and informs it of the condition and content of the commitment, as well as a timeout set for the debtor to accept the commitment and the condition to become true. Then the monitor agent creates an *unset* commitment and proposes the commitment to the debtor that can accept or refuse the commitment, leading to a *pending* or a *cancelled* commitment state respectively. In *pending* state the condition, represented as a temporal proposition, for the commitment to become *active* is evaluated. If the commitment's condition formula evaluates to a true before a certain timeout the commitment becomes *active* and the condition of the commitment, another temporal proposition, is the next formula to be evaluated. There is a deadline for the commitment when it has its state changed to *fulfilled* or *violated* and it starts counting from the moment its state changes to *active*.

The operations over the commitments are performed by a specific agent in the Institutional Environment (the Monitor Agent specified in Section 3.7.2). They can be triggered due to a message coming from a creditor, requesting a monitored interaction, a debtor accepting or refusing a commitment proposal. The Institution Agent can as well send messages that trigger operations on commitments due to the results of the evaluation of the condition and content formulas in the model checker present in it. This will be better explored later on this chapter.

The operations that can be executed over the commitment objects and change their state, as identified in Colombetti, Fornara and Verdicchio (2002), are:

- Make commitment: this operation instantiates a commitment object, creates an identifier for it and sets its state to *unset*.

- Set commitment: this operation changes the state of a commitment.

- Cancel: this operation cancels the commitment due to a message from the debtor that refuses to accept a commitment or due to a message from the Institution Agent that is unregistering an agent from the Institutional Environment.

- Release: this operation discharges a debtor from a commitment. It is executed due to a request from the creditor of a commitment, or due to a message from the Institution Agent that is unregistering an agent from the Institutional Environment.

- Discharge: used when the debtor fulfils the commitment. It is executed due to a message from the Institution Agent saying that the formula that represents the content of a commitment was evaluated to true, before the deadline set to it.

- Assign: Replace the debtor of a commitment in the case of an agent is changing roles in the institutional environment.

3.5. Reputation in the Institutional Environment

The commitment management mechanisms from the Institutional Environment together with the reputation management mechanisms are what characterize our environment as a normative system. The commitments take care of what the agent should do in the institutional environment to behave according to an agreement accepted during registration in the institutional environment. The obedience to the norms of the system will reflect in the reputation of the agents and this will determine if the agents will be rewarded or punished by the institutional environment.

The rewarding and punishment of the agents has no general rule. It depends on the institutional environment architect and on the application domain. The only rule that is put out there for the agents is that certain interactions are compulsorily monitored and others not. The agents can make use of the reputation mechanisms to choose with whom they will interact in the future and the institutional environment can choose what to do with agents that have good or bad reputations.

Reputation encourages the observation of the norms represented by the commitments in the Institutional Environment and delivers ways to grant openness to the multi-agent systems implemented using our approach. The life cycle of commitments is managed in the institutional environment and the fulfilment or the violation of the commitments triggers

the reputation update of the agents registered with the institution. If a commitment is *violated* the debtor agent is punished losing reputation points and when a commitment is *fulfilled* the debtor agent is rewarded with reputation points by the reputation agent.

Any agent initiator of a conversation can check the reputation of other agents before the start of the interaction and choose not to interact with that agent. In that way agents that behave badly may be asked to be participants in a conversation by initiator agents at a very low frequency, or not at all. It will still be asked to participate in conversations by agents that do not check the reputation prior to interaction but that is the choice of the interaction initiator and it adds to the open characteristics of the institutional environment and freedom of speech of the agents registered to the institutional environment. The responder can also check the reputation of the initiator before accepting further messages from the initiator.

On reputation update, when a commitment reaches one of its final states of *fulfilled* or *violated*, the reputation agent informs the external agent that it will be rewarded or punished, and asks the institution agent to punish or reward the agent in the institutional environment. The only punishment modelled by the system is the expelling order, which is applied by the institution agent. Other rewards and punishments are difficult to predict because they are strictly related to the application domain that the institutional environment models. Therefore, this is left to the architect of the system, and it will be done in the institutional agent because it has the registry of agents that are currently in the institution or that were there and might come back, and the model checks the updates the states of the commitments.

The external agents are most often modelled and built by other architects than the one that built the institutional environment. Therefore, they have their own way to judge and use the reputation mechanisms of the system. The reputation mechanisms are implemented to offer this feature in a way that the system architect can implement the necessary policies regarding reputation, that is, reward and punishment, requested by the application domain, and for the external agents to have access to this information and use it to meet their objectives.

3.6. Specification of the Institutional Environments.

A significant characteristic of our approach is its open and distributed nature. Elements are organised in such a way that they are not compelled to report their actions to any other participating agent in the structure. We define our Institutional Environment (IE) as:

$$IE = (O_s, N_s, I_a) \quad (1)$$

Where:

– O_s stands for system ontology;

– N_s stands for normative space;

– I_a stands for institutional actions.

The N_s is defined as:

$$N_s = (R_s, R_e, O_c, L, C) \quad (2)$$

Where:

– R_s stands for system roles;

– R_e stands for external roles and corresponds to roles available to be played by external agents that register to the institutional environment; they are made available in the form of Deputy Agents that are allocated to external agents that will fulfil the role in the institutional environment;

– O_c stands for context ontology;

– L stands for content language and represents the language expressed in the content attribute of the FIPA ACL messages exchanged among agents in the N_s.

– C stands for commitments. These can be social commitments when an external agent commits with the institutional environment represented by the Institution Agent, or compulsory commitments embedded in the interaction protocols used by the external agents. Regular commitments that are made necessary through the request of monitored

interactions not predicted in the interaction protocols are not listed here but most definitely can appear in the institutional environment.

We identify three types of commitments:

- The social commitment is defined with an external agent as debtor and the Institution Agent as creditor.
- The Institution Agent represents the entity in charge of managing the IE, and therefore rules from the society are enforced through the creation of the social commitments and their observation when defining the reputation of the agents that are registered in the IE. Compulsory commitments are defined by the context of the application domain, and their creation is embedded in the CPNs of the Deputy Agents. Deputy Agents enact the roles that external agents can play in the IE, and the compulsory commitments are defined among external agents.
- Regular commitments are the ones that are not defined at design time but that can become necessary in some conversations at run time. As the external agents have the option of developing conversations that are not pre-defined by the IE, they might find it necessary to create a commitment that was not predicted at design time.

The O_s and O_c are defined separately because they represent concepts related to the management of the IE, in the first case, and concepts related with the application domain where the IE is found in the second case. These represent separate concerns.

The content language L is used to construct expressions that contain elements from O_s and O_c, and represent statements exchanged among agents in the IE through FIPA ACL messages.

The system level infrastructure identifies system agents that assume roles from R_s and are deployed to manage the interactions in the N_s. The following describes such agents:

- Institution Agent: The first step for an agent to get into the institution is to register with the Institution Agent and assume a role in the artificial society. Upon registering, the agent will gain access to the N_s in the form of a workflow that represents a role in the IE. This is implemented in the form of a CPN that represents a suggested path or course of actions the recently registered agent should follow to

obey the norms that constrains the institutional environment; these are made available in the form of the Deputy Agents and the access to its CPN through the external agent developer. The N_s can be as constrained as the institutional environment developer wants; it depends on the degree of restrictions, security or access he wants to deploy in the system, for example, demanding the use of monitored interactions for certain activities. It is important to mention here that the member agents will have their own goals and strategies to obtain them, and even though they have knowledge about the workflow of the institutional environment they still can participate freely in conversations without compulsorily following it, through unmonitored interactions. The Institution Agent also has a Model Checker implemented that evaluates formulas that represent temporal propositions that define the condition and content of commitments. This will be better explained later on in this chapter.

- Monitor Agent: This system agent represents a monitoring authority in the institutional environment. It monitors certain activities, defined as institutional actions, according to the norms defined in the normative space. The monitoring is done through the use of commitments and observation by the monitor agent of the commitments' life cycle. Once in the artificial society, the agent can commit itself with other agents to perform tasks and propose to other agents to assume commitments to perform tasks for it. In case the agent does not have an acceptable level of trust in the agent with which it is starting the interoperation process, it can ask for a monitored interaction, where the monitor agent will audit the commitment shared by the agents engaged in interaction. That interaction is stored in a database of audited interactions for later examination, if requested to the monitor agent. It is important to mention here that the system agents will always follow the N_s and with that the norms that regulate the institutional environment. But again, knowing the rules, some groups of agents might be formed without the use of any monitoring in their interaction only based on the trust they developed in each other based on their own history.

- Reputation Agent: The agents in the open MAS have access to a system agent called the Reputation Agent. This agent is responsible for giving information about other agents that have been present at some time in the society and have developed a reputation. The agents are not obliged to report nor consult the Reputation Agent

prior to every interoperation they take part in. This offers flexibility and the possibility of open MAS implementations with verifying degrees of agent autonomy. This agent will update the reputation of the debtor agent when the life cycle of the commitment ends.

An Institutional Environment usually is built with the intent to motivate the agents to cooperate with the norms of the system. In the Normative Space the reputation that the agents build in the system can be used to motivate them to behave in accordance with the system norms, or even other agents' beliefs. That pursuit for good reputation is seen as a motivation for them to cooperate in the system or deal with sanctions and punishments. More details on this motivation can be found in Sections 3.6.3 and 3.10.

3.6.1. Agent Roles and the Conversation Space

In our approach CPNs are used to represent agent roles and agents' conversations. In that way we visualize an agent's role as part of a CPN that describes the messages exchanged by a specific group of agents in the institution. The overall institution conversations are represented as a CPN that has specific access points, where smaller CPNs, representing roles of individual participating agents, can be plugged in. Therefore the construction of the environment is the connection of lower level CPNs forming a higher level CPN (Nowostawski, 2009) (Jensen, 1992). Details about the CPN model used are given in Section 4.4. With that approach an agent will be involved in the part of the conversation appropriate for its role in the institution.

The access points are special CPN places present in all roles defined in the Institutional Environment. They are called "In" and "Out", where incoming and outgoing tokens are put. In our approach these places are used to store FIPA ACL Messages that are exchanged among the agents in the Institutional Environment. Therefore all the agents are conected via a FIPA message passing layer.

The CPNs that represent workflows defining roles played by external agents in the institutional environment are implemented as Deputy Agents. These are agents that implement all the generic institutional acts that are used for the communication and management of the institutional environment, as well as the specific institutional acts that

are identified in the application domain for each role played by external agents defined in the institutional environment.

The Deputy Agents could be seen as *organizational proxy agents* (Hübner et al., 2010), since they do not have autonomy in what they do, even though they could have if we judge it necessary in the future. They manage the interaction protocols modelled that comprise the workflow for a role identified in the application domain. They model all the institutional acts defined for the system and for the application domain related to a specific role, as well as the possibility of using unmonitored interactions among agents inside the institutional environment, thereby making it more scalable – see details in Section 3.10. It is necessary to mention that we did not develop a middleware, such as AMELI (Esteva et al., 2004), with proxy agents such as the *governors* (Esteva et al., 2001), which uniquely develop: interactions predicted at design time that are translated into interaction protocols represented in Finite State Machines (FSM). See more details on organizational proxy agents in Section 2.1.

The Deputy Agents manage CPNs that are connected through a FIPA message passing layer, and they do not live inside middleware as in other approaches, such as AMELI (Esteva et al., 2004), S-Moise+ (Hübner, Sichman and Boissier, 2006) and TeamCore (Pynadath and & Tambe, 2003). It was a software engineering decision to bring the access to CPNs to the agents external to the institutional environment through the Deputy Agents. When modelling a role for the institutional environment that will be played by external agents, the institutional environment designer will have available to start with a CPN with the institutional acts related to the management of the institutional environment, which leads to conversations with system agents. The role designer will add the logic dealing with the other institutional acts needed in the application domain.

The CPNs of the Deputy Agents are made available to designers of external agents so that they can know the possible conversations their agents can take part in the institutional environment. The deputy agents can run locally where the external agent is running and communicate with the other agents in the institutional environment through FIPA ACL Messages. They implement the protocols the agent can participate when playing a role defined in the institutional environment.

Every agent CPN in the institutional environment will have a local state that concerns its role in the whole system, modelled as a set of CPN tokens. By distributing the state

representation among a set of tokens, it is easier to represent the local state of an individual agent role (in the context of a larger institution), and this can be useful for local management of individual agents.

We use the Otago Agent Platform (Opal) (Nowostawski, Purvis and Cranefield, 2002) framework for lower level FIPA communication services and the JFern CPN Tool Box (Nowostawski, 2009) for CPN creation, simulation and deployment. The Opal architecture allows for the development of agent templates as message processors that implement patterns of behaviour. These patterns of behaviour were extended in our implementation as calls to the JFern CPN Simulator to add and subtract tokens from the CPNs that represent the roles of agents.

3.6.2. Commitments

In our approach a commitment is an object created at the agent level and handled by the Monitor Agent involved in the interaction. The Debtor is the agent that makes the commitment, and the Creditor is the agent relative to which the commitment is made.

We have adapted a model defined in Colombetti, Fornara and Verdicchio (2002) to our needs. The commitment states used are: *unset*, *pending*, *cancelled*, *active*, *violated* and *fulfilled*.

The Monitor Agent is the system agent that manages the commitments in the open MAS. We have defined a CPN that represents the monitor role and manages the commitment object from its instantiation to its storage in the commitment log: a database of commitments that have reached one of their final states (*cancelled*, *violated* or *fulfilled*).

The Monitor Agent will deal with monitored interactions in the normative space. Operations on commitments reflect in its state and together with the norms defined in the normative space a monitored agent can have a change in its system reputation or in an extreme case it could be banished from the institutional environment, depending on the constraints imposed by the application domain.

The audit process performed by the Monitor Agent can be visualised as the proof given by the debtor agent that he performed some task. That would have the form of a commitment object with its state equal to “fulfilled”, acknowledging some information about a task the

debtor should perform. Basically, the debtor would be committing itself formally to an authority in the open MAS (represented by the Monitor Agent) that it did something. If an agent commits too often to false statements, the Monitor Agent should receive many complaints about that agent, which would lead the Monitor Agent to use its power to update the reputation of agents in the open MAS.

The normative space guides the Monitor Agent during the monitoring process; its role is a commitment management protocol that manages commitment objects according to their life cycle. Participating agents can decide to break the commitment but the normative space defines the level of tolerance to those actions, and depending on the application domain that the institutional environment is implementing, an agent's acts can decrease its reputation to such a level that other agents that consult the Reputation Agent will cease to communicate with the untrustworthy agent. The time factor is an important element in the representation of commitments. To express a commitment formally, it is necessary to find a representation mechanism able to handle the constraints of time found in the definition of commitments. We represent the idea of commitment's timeout and deadline as in Colombetti, Fornara and Verdicchio (2002). The timeout is related to the period of time available for the commitment to become *active*, and the deadline is related to the amount of time available for the commitment to be *fulfilled*.

Another aspect to be analysed is the temporal characteristic of the condition and content of the commitments. To comply with that in our approach we use formulas that represent temporal propositions to define the condition and content of the commitments. These formulas are evaluated against a model that defines states where propositions about the conversations that take place in the institutional environment are true or false. The formulas are evaluated against the model in a Model Checker implemented in the Institution Agent.

The state of the Monitor Agent is maintained by the management of the CPN that models this agent's workflow. The commitment objects have their own life cycle, as defined in Section 3.4 earlier in this chapter, and it defines the states that the commitment objects can acquire. The model checker maintained by the Institution Agent defines a model for the institutional environment against which temporal propositions from the commitment objects are checked. The insertion of new states in the model defined in the model checker change the outcome of the evaluation of the temporal propositions and consequently change the state of the commitments. There is a very important channel of communication between

the Institution Agent and the Monitor Agent to update the state of the commitment objects (see Figures 3.3 and 3.4).

3.6.3. Reputation Influence in the Institutional Environment

All the requisite institutional information is given to the agents upon their registration in the institutional environment, and will be available from the Institution Agent for consultation at any time. However, the Institution Agent does not maintain any control over the registered agents. In fact it is the reputation that the agent has in the institutional environment that will determine its useful existence in the system. Based on the trust the agents have for each other, they will interact or not. With time and the development of a number of interactions, agents can build up trust networks and establish trust relationships with each other.

The reputation the agent develops in the institutional environment together with restrictions of the normative space defined in the N_s, may restrict levels of access to certain resources. Concepts such as level of access to resources and services can be modelled in the normative space through the context ontology O_c and implemented in the N_s. Therefore, as all the agents are aware of the rules of the institutional environment they know that their actions might not only affect their reputation but diminish their level of access to resources in the system.

We define a trust relationship ontology that defines the concept of trust and reputation and how they influence the system agents and can be used by external agents to compose their own definition of trust. That ontology is available for the agents that are joining the institutional environment to understand the information managed by the Reputation Agent. The reputation update model sometimes needs to express characteristics of the context in which the institution was developed, and different reputation update models can be attached to the Reputation Agent CPN to express that.

Another important aspect of that approach is that the external agents can have their own definition of trust and use the one defined by the institution to add information to it, or simple ignore it. The agents are not compelled to use the Reputation Agent before every transaction. They can have, for example, a history of their conversations internally and their own information about other agents and choose to refuse certain kind of interactions from

some agents that they might not trust, but in case they are willing to use the information agent, it is available. Being aware of the N_s they can calculate the risk of losing privileges in the institutional environment and act as they will.

Our intent is to analyse the use of commitments to manage the social order in the institutional environment. There is a vast universe of ways to calculate the reputation score of an agent in an open MAS and it is a future work of this thesis to analyse the use of alternative approaches to refine the social order mechanism implemented here. As it was an open area of work we were concerned with making the Reputation Agent CPN extendible and adaptable to the implantation of different ways to calculate the reputation score. This will be demonstrated later in this chapter.

3.7. Workflows for the Institutional Environment

In this Section we describe the CPNs that represent the workflows of all the agents that are present in the Institutional Environment.

3.7.1. Institution Agent Workflow

The Institution Agent is the system agent responsible for the registration, unregistration and change of role of agents external to the institutional environment. All the interactions between the external agent and the agents in the institutional environment are made upon the exchange of FIPA ACL messages. The FIPA ACL messages will have in their content reference to institutional acts identified in the institutional environment application domain when modelling it. For example, in an auction institutional environment we could have the following contents: (register (role auctioneer)), (bid (value 1000)) or (declare_winner (agent_name buyer01@auctionhouse).

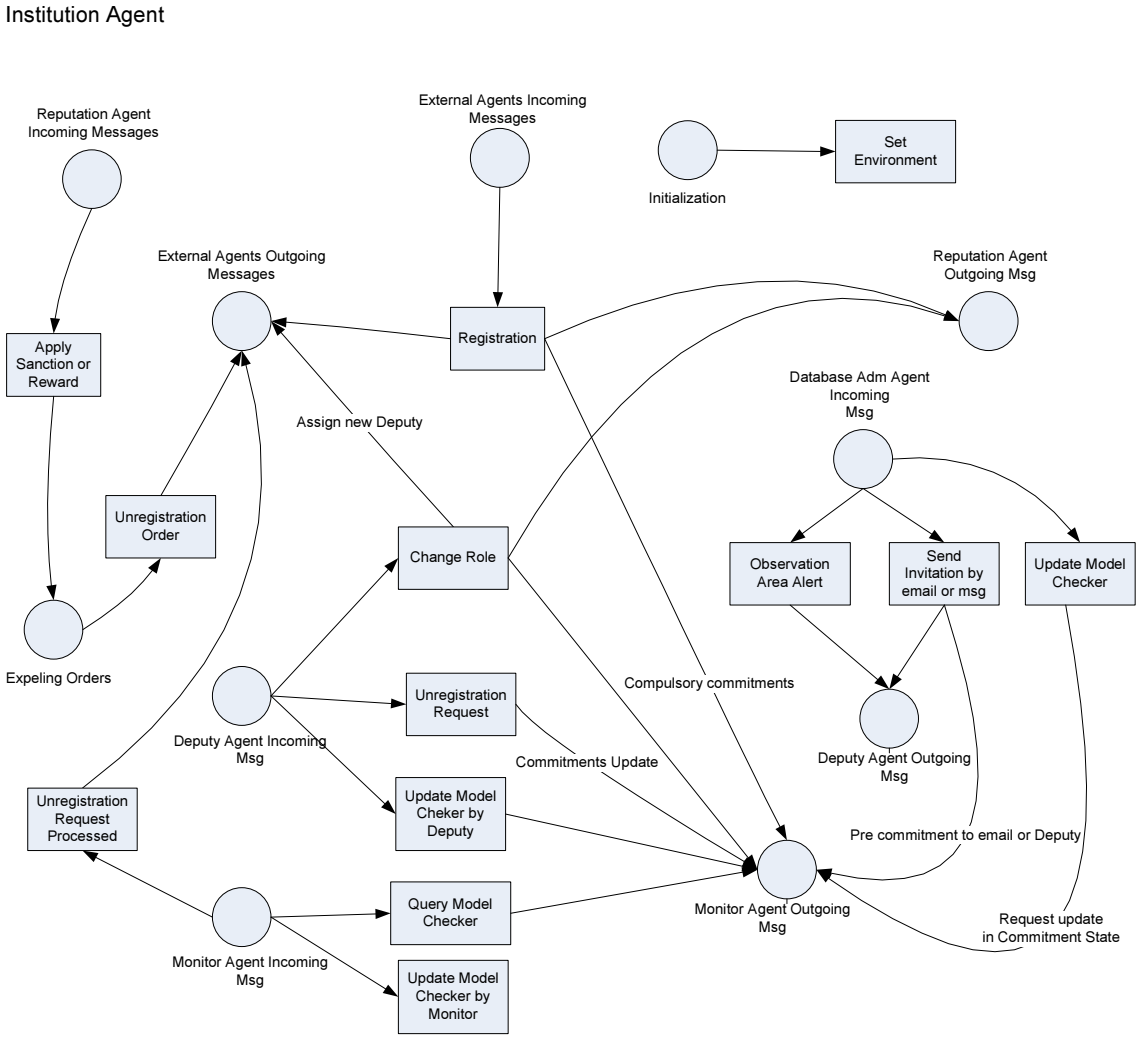


Figure 3.3: Institution Agent Workflow

Like the other system agents and all the deputies modelled, the Institution Agent has its role represented by a CPN, shown in Figure 3.3 above. One can identify the In places, where incoming messages from different agents arrive represented as CPN tokens of type message), and Out places where messages are put to be sent to other agents. It is important to mention that the CPNs are connected through FIPA message passing; therefore there is no problem to have all the CPNs that represent the agents´roles with these special places named In and Out. The routing of messages is made through Opal and, for example, all the messages addressed to the Institutional Agent will be delivered to its In place without causing any confusion with the other In places from other roles. The Out place, likewise, is always checked for the arrival of tokens every time a run in the JFern CPN simulator is made; if there are tokens available they are taken out of the Out places and delivered to the

In places of the other agent through Opal FIPA message passing layer. The workflows are therefore connected through Opal´s FIPA message passing layer[4].

In the case of the Institutional Agent, it receives messages from the external agents, reputation agent, deputy agents and monitor agent. The external agents send requests to register with the institutional environment, the institution agent checks with the reputation agent if the email, or some other form of unique identifier predefined at design time, that is used by the external agent was registered before in the institution. In case it was, the institution agent can refuse new registration due to low reputation recorded before in the institutional environment. In case the institutional agent decides to register the external agent, it answers to the request saying the roles available to be played by the external agent in the institutional environment. The external agent can then request the registration to play a determined role.

The records of reputation of an external agent will influence the new registration of that agent in the institution depending on how the institutional environment designer decides to apply sanctions to agents with low reputation. The Institution Agent then replies saying if the request was accepted, and in the case of acceptance a deputy agent is allocated to that external agent.

The deputy agent is modelled in such a way that the external agent developer will know all the institutional acts pre-identified in the institutional environment by the institutional environment designer; they are represented by the transitions in the workflow CPNs. The workflow CPN can be analyzed by the external agent developer so that he can understand how the agents will deal with these institutional acts and if they lead to the generation of messages to other agents[5].

Any commitments that the external agent has to assume as debtor of the institutional environment are created in the “Registration” transition. These commitments are compulsory for the registration to be accepted by the Institution Agent and they represent social norms that the external agent assumes with the institution. These are identified as general rules of behaviour in the institutional environment, for example, an agent must

[4] This will be better explored in the next chapter.
[5] An automated way to read the deputy agent workflow in order to determine all the pre-defined institutional acts, which agents uses and to which agents they are used, could be implemented via a procedure that reads the XML formalization of JFern defined CPNs. That feature is out of the scope of this thesis but is a possible topic for future work.

commit to not reporting false information in an institutional environment that models collaborative systems (De Oliveira et al., 2009). There are ways to verify this commitment via testimony of other agents, for example, as we can verify in Chapter 5. A message is prepared by the “Registration” transition and is put in the Out place to the Monitor Agent as a request to generate a commitment that represents the social norm.

The Institution Agent has access to the Agents Registry, which is a data structure or a connection to a database depending of the projection of the number of possible registrations at design time. This database or data structure is accessed as a side effect from the transitions “Registration”, “Unregistration Request Processed” and “Send Invitation by Email or Msg”

The Institutional Agent has access to an interpreter that runs a model checker that can evaluate social expectations (Cranefield and Winikoff, 2011). From this model checker we use only its feature of LTL Temporal Logic formulas evaluation against a model. The model defines a history of states where propositions are true or false. The commitment objects, which are managed by the Monitor Agent, have as condition and content temporal logic formulas that are checked by this model checker. The facts that change the truth value of the propositions present in the model are inserted in the interpreter as side effects from the transitions “Update Model Checker by Deputy” and “Update Model Checker by Monitor” and “Update Model Checker by Database Administrator Agent”. These updates will depend on the interaction of the Institution Agent with the Deputies and the Monitor Agent and Database Administrator Agent.The effect of updating the Model Checker is the receiving of information by the Checker. That information will effectively update the model present in the checker. More details on the model checker used can be found in Sections 2.7 and 4.5.

The Deputy Agents can send messages to the Institutional Agent in the case of request for unregistration with the institutional environment, request for changing of roles in the institutional environment, and update of the model checker, which is represented by the dynamic model in the model checker.

The invitation to an external agent can generate the creation of a pre-commitment with the monitor agent. The pre-commitment, which is in fact a commitment object in state *unset*, will become a commitment if the agent that is being invited accepts the invitation. This mechanism is implemented to reach the agents that are registered with the institutional

environment and the agents that might register in the future using the email specified by the agent that is sending the invitation. The commitment will have as creditor the inviter or the institution agent, and as debtor the invited. All this process is done in that way because the agent that is sending the invitation may want to propose to the agent that is receiving the invitation, that it can possibly assume a commitment with the inviter or with the institution, in this case a social norm.

The “Change Role” transition will verify if the requesting agent can assume the requested role; this is done according to predefined rules for role change. These rules are represented as a simple table that states to which role an agent can change coming from other roles; they are defined in this transition by the institutional environment designer. If the change of roles is allowed the institution agent will unregister the agent that is requesting the change of roles, ending the current deputy agent assignment, and then register the agent again assigning a new deputy agent that has the workflow of the requested role.

The “Update Model Checker by Deputy” transition receives messages from the deputy agent that will demand the insertion of facts in the model checker to create a new state in the model checker´s history. This will affect the evaluation of the formulas that represent temporal propositions modelling the condition and content of commitments. Every time a fact is inserted in the model checker the formulas are evaluated and the results are sent to the monitor agent, so that it can update the state of the commitment objects.

The Monitor Agent can send messages to the Institution Agent requesting for the update of the model checker by the insertion of new formulas that represent the condition and content of newly instantiated commitment objects, or simply querying the model checker about the truth value of some formulas. In this last case, the answer would update the state of the same commitment object managed by the Monitor Agent.

In the figures that represent workflows in this chapter, the labelling of the output arcs give a brief description of the tokens that are being consumed from an input place or produced by the transitions. For example, in the case of Incoming and Outgoing places the arcs describe what will be the purpose of the messages consumed or added to these places respectively, since the tokens in these places are FIPA messages. Arc annotations (see appendix A) ensure that exactly one arc has a token generated based on the examination of the message taken in the In place. Arc labels may be suppressed if their meaning is clear from the transition they go to.

In the CPNs that define the workflows, the existence of many In and Out places is justified because of an extension made to the Opal Agent Platform to handle these workflows as behaviours of agents. The FIPA messages are filtered and addressed at the Opal level. If we did not have that feature from Opal we would have only one In place and this filtering and routing of messages would need to be done by the annotations of the input arcs coming out from the In place. We mention here, again, that the workflows are instantiated in separate simulators, one for each agent, and the CPNs are connected via FIPA message passing implemented by the Opal agent platform.

The Reputation Agent incoming messages place will receive messages for the answering of a reputation query done by the Institution Agent, and with possible requests for the application of sanctions to some agent that has reached some predefined low reputation score in the institutional environment. These sanctions will depend on how the institutional environment designer wants to punish a badly behaving agent in the institution. That could vary from a simple warning to the expulsion of the agent from the institutional environment.

All the In Places from the workflow can receive FIPA ACL messages asynchronously, represented by the addition of tokens. Those messages will represent predefined institutional acts that represent message patterns defining commands coming from the expected sender agents. On the arrival of the messages the CPN that represents the workflow runs until there are no more transitions to be fired. When this final state is reached all the messages that are present in the Out places are collected and sent to the respective recipients. All that is implemented using the Otago Agent Platform (Opal) for FIPA ACL messaging exchange and JFern CPN Simulator (Nowostawski, 2009) for the running of the workflows.

Some method calls are present in the action statements of the transitions of the CPNs that represent the workflow. These methods access certain operations relevant to the management of the institutional environment and are intended to be independent of application domain. All the methods are called on the agent_owner variable which is a reference to the agent that owns the workflow. In particular methods for updating the model checker and setting deadlines for propositions are present in the transitions of the Institution Agent (see Figure 3.3).

3.7.2. Monitor Agent Workflow

The Monitor Agent is the system agent responsible for the management of the life cycle of the commitment objects that are instantiated in the institutional environment. It has access to a data structure that stores the commitment objects, accessing it constantly for update of the commitment's state.

The Monitor Agent Workflow has In places for the Deputy Agents and the Institution Agent. The Deputy Agent can propose a commitment, which will trigger the "Make Commitment Unset" transition. This transition will generate a message to other deputy agents with the proposal of the receiver accepting a commitment as a debtor. The Deputy can accept or refuse the proposal and that will be dealt with by the "Set Commitment to Pending or Cancelled" transition.

If the commitment is set to *pending* a message updating the model checker in the Institution Agent is prepared, sending the formulas for the condition and content of the new commitment. The Deputy Agent can also set the commitment to *cancelled* or release a debtor from a commitment directly. These operations will change the state of the commitment to *cancelled*.

The Institution agent will have a great deal of interaction with the Monitor Agent because the first has the model checker that contains the propositions that are evaluated by the formulas that represent the condition and content of the commitments that the Monitor Agent manages. Therefore, the Institution Agent can propose a commitment to the Deputy Agent, which is seen as a social norm or a commitment with the institutional environment.

The Institution Agent can also set a commitment's state to *cancelled* due to a timeout. Timeouts are set to determine a waiting time for the commitment change state from *pending* to *active*. We have them set at the Monitor Agent clock at the moment the commitment becomes *pending*. The timeouts generate events that change the state of the commitment to *cancelled* if they expire and send a message to the Institution Agent to update the model checker history with a state where this fact is true. It is not compulsory to set a timeout for every commitment object instantiated.

The Institution Agent can set the commitment's state to *active* if the formula that represents the commitment condition is evaluated to true, and it can also set a commitment's state to

fulfilled or *violated* according to the evaluation by the model checker of the formula that represents the content of a commitment.

The update of the state of a commitment to *fulfilled* or *violated* will generate a message to the Reputation Agent requesting the update of the reputation of the debtor of the commitment positively in the case of commitment fulfilment, or negatively in the case of commitment violation.

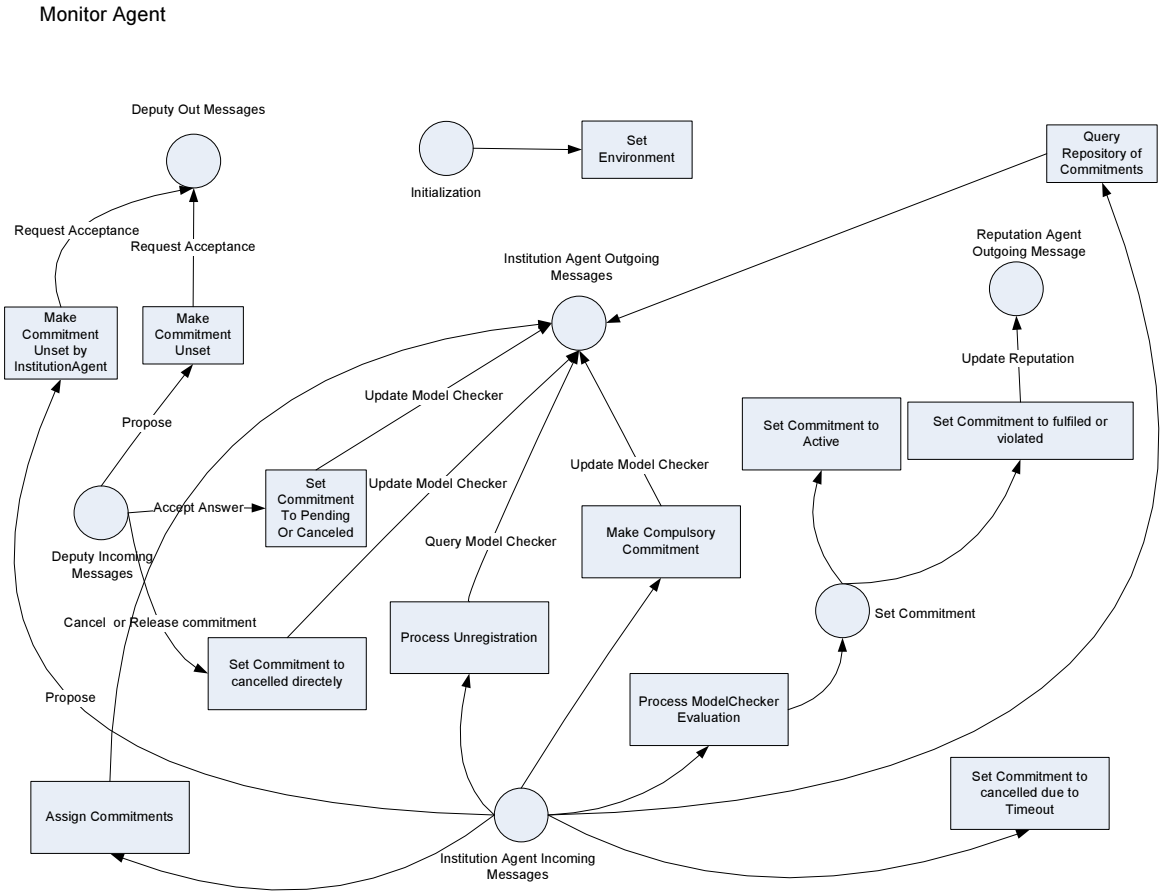


Figure 3.4: Monitor Agent Workflow

The Institution Agent can also query the Monitor Agent for information on a specific commitment. This will trigger the “Query Repository of Commitments” transition that will query the data structure that stores the commitments of the institutional environment, and generate a reply message with the requested information. Possible queries are, for example, commitments from a creditor, commitments from a debtor, *cancelled* commitments, active commitments and so forth.

3.7.3. Reputation Agent Workflow

The Reputation Agent manages the reputation of all the external agents that have or had at any time a deputy assigned to it by the Institution Agent on registration with the Institutional Environment. It manages a data structure for the External Agents' reputations and receives messages from the Deputy Agents, Monitor Agent and Institution Agent.

The Deputy Agent can query the Reputation Agent about the reputation of any agent that is or was at some point registered with the institutional environment. This will generate a reply message via the transition "Query Process".

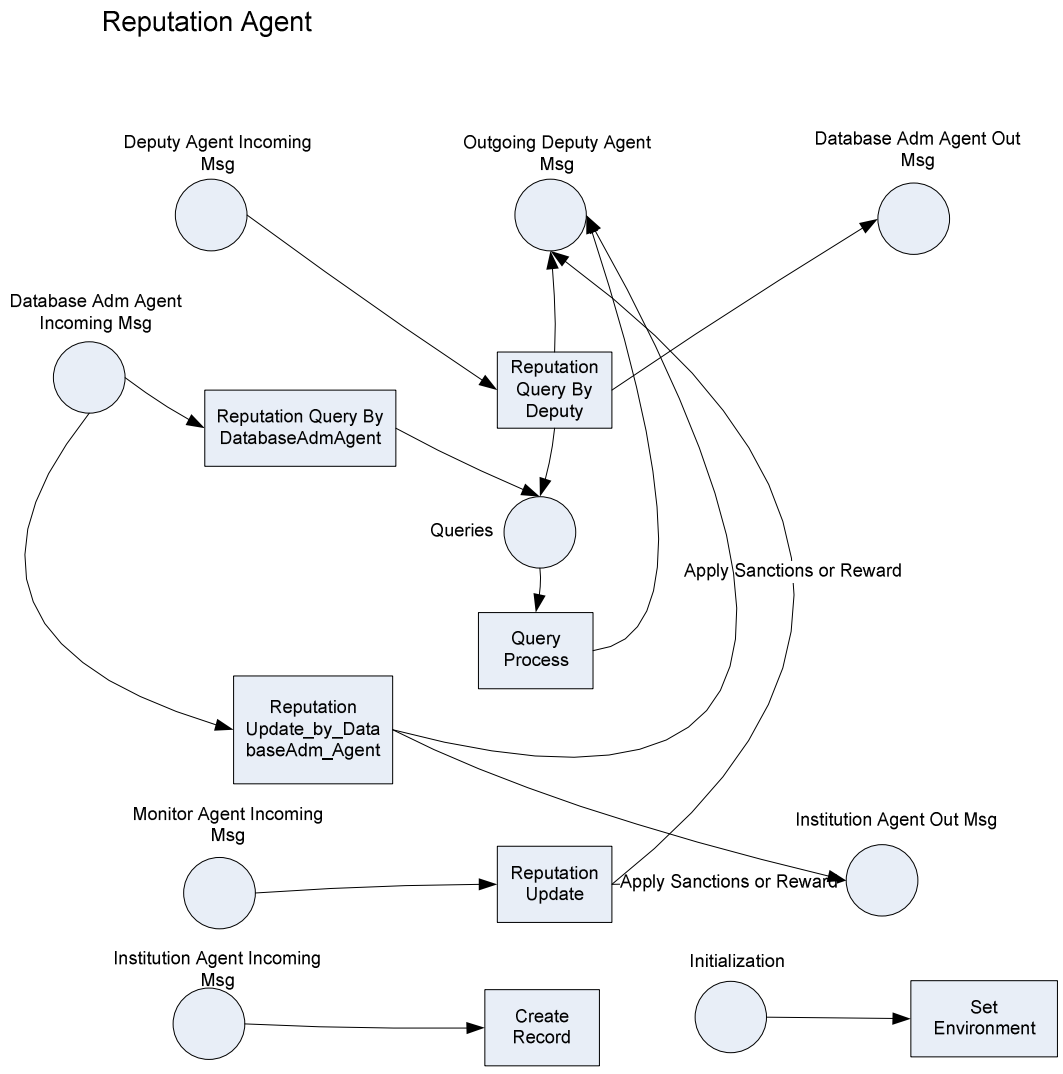


Figure 3.5: Reputation Agent Workflow

The Monitor Agent can send a message to the reputation agent requesting an update to the reputation of an agent due to the evaluation of the content of a commitment to *violated* or *fulfilled*. At some stage defined by the institutional environment designer, sanctions or rewards might be applied or awarded to the deputy agent that is a debtor of a commitment. The messages to request these actions are sent to the Institution Agent and a message informing the Deputy Agent of the reward or sanction is sent then.

The Institution Agent can request the Reputation Agent to generate a record for an External Agent that is registering with the institutional environment. Observe that this request does not generate a reply message and the transition “Create Record” does not produce any output token, it only accesses the data structure that stores the reputation of the agents and is part of the internals of the Reputation Agent. This direct access to the internals of the Reputation Agent is implemented as an action statement of the CPN transition, which represents a side effect its evaluation.

3.7.4. Database Administrator Agent Workflow

The Database Administrator Agent Workflow represents the role of a system agent present in institutional environments that have the necessity of manipulating databases. This role is necessary to diminish the responsibilities of the Institution Agent regarding database transactions. Therefore, if the institutional environment that is being designed has, for any reason, data that needs to be managed by a Database Management System it will be necessary to identify institutional acts that will represent interactions with the Database Management System.

The workflow of the Database Administrator Agent defines interactions with the Deputy Agents, Institution Agent, Monitor Agent and Reputation Agent. There is an In place for the Deputy Agents to trigger the “Query Database” and “Comment the System” transitions. These are the generic institutional acts identified, meaning that the deputy agent can at least query the database, for any reason, and make comments about the institutional environment.

The “Specific Behaviour” transition represents all the other institutional acts identified in the application domain in which the institutional environment is implemented. This transition will expand to another CPN that will have as input and output places the ones

present in Figure 3.6. Therefore, the Database Administrator Agent will interact with the other system agents as well, but again this depends on the institutional acts identified at design time.

The transitions present in the workflow represented by Figure 3.6 have action statement side effects that implement the transactions with the Database Management System. These transactions are mainly for the storing and retrieving of data stored in the Database Management System that can only be identified when working on the application domain to which the institutional environment belongs.

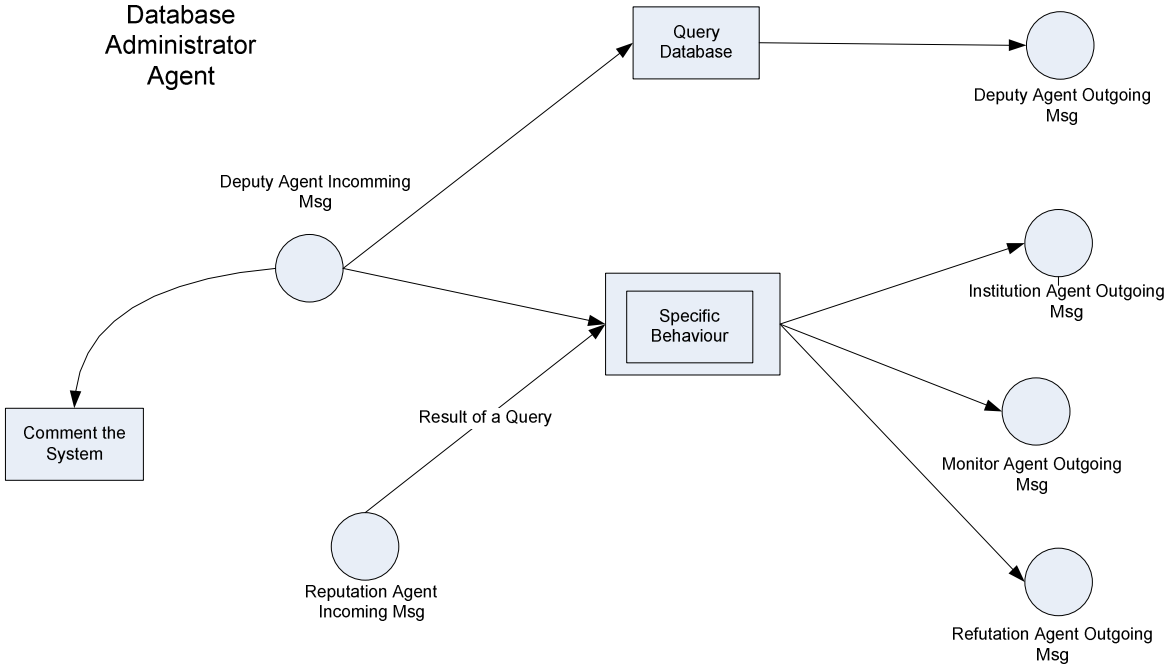


Figure 3.6: Database Administrator Agent Workflow

3.7.5. Deputy Agent Workflow

The Deputy Agent is responsible for intermediating the conversations between the External Agents, registered with the institutional environment, and the System Agents. It intermediates conversations as well between External Agents that are registered with the institutional environment, and that are currently online.

The Deputy Agent Generic Workflow models all the possible use of institutional acts identified in the conversations between the External Agent and the System Agents in the institutional environment. The CPN that represents this workflow has a special transition,

called “Process Specific Institutional Acts”, which models the specific behaviour of the agent regarding the role it plays in the institutional environment. This specific behaviour is modelled and added to the generic workflow putting together the institutional acts that depend on the application domain and the ones that represent the functioning of the institutional environment in general.

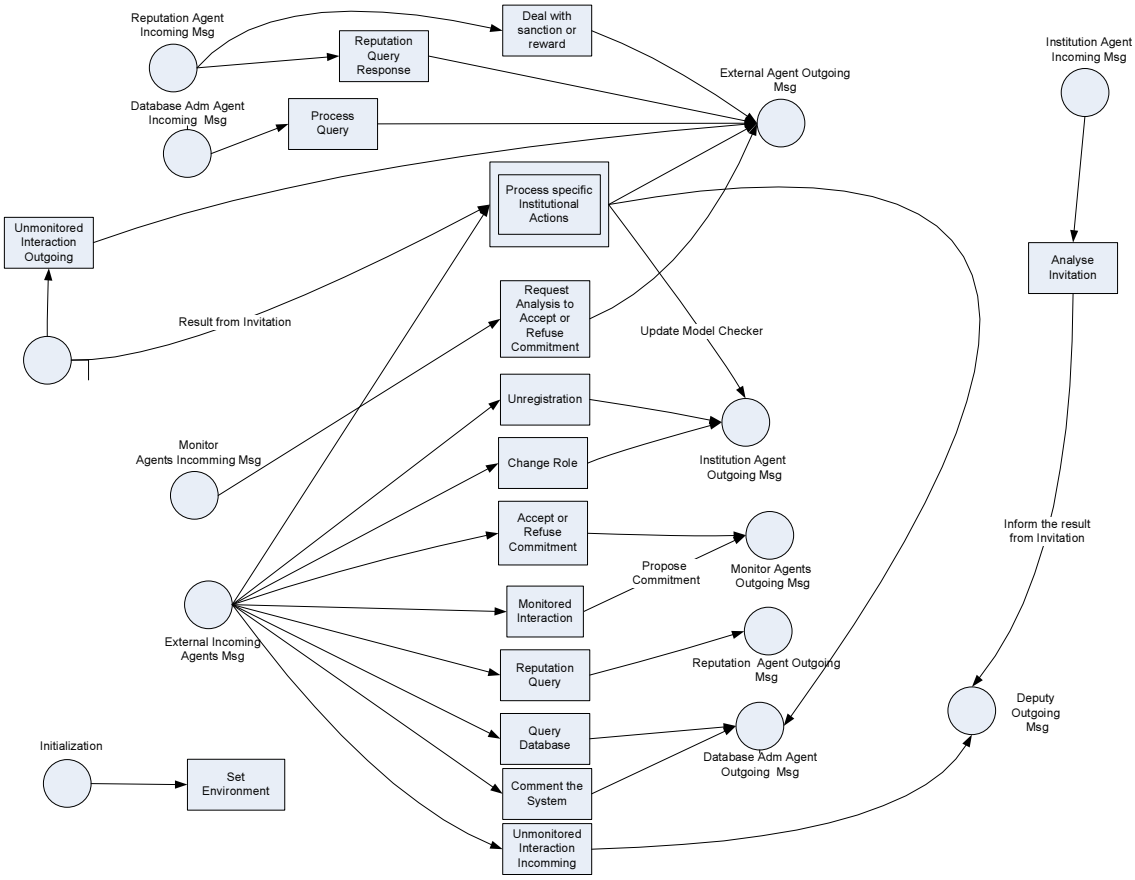


Figure 3.7: Deputy Agent Generic Workflow

The External Agent can send messages to its Deputy Agent using the generic Institutional Acts to:

- Unregister with the institutional environment, which will generate a message to the Institutional Agent;
- Ask the Monitor Agent for a Monitored Interaction, which will trigger the process of generating a commitment object by the Monitor Agent and the respective proposal of a commitment to a debtor;

- Query the Reputation Database, stored in the Reputation Agent, for the reputation of an agent that is or was registered with the institutional Environment.

- Ask the Institution Agent to change the external agent's role in the institutional environment;

- Query the database managed by the Database Administrator Agent;

- Comment on the institutional environment, which will generate a message to the Database Administrator Agent.

An External Agent can also use institutional acts that were identified in the application domain where the institutional environment was built. These are dealt with by the expansion of the transition "Process Specific Institutional Actions". This subject will be better explored in Chapter Five where we discuss WikiCrimes Institutional Environment (Furtado et al., 2010).

There is the possibility of starting conversations among external agents in an unmonitored mode. This means that the external agents in the institutional environment have what we call freedom of speech, so to speak. They can start conversations depending on their internal states and with all the responsibility of understanding each other. The institutional environment would be the meeting place for them.

The External Agents might also do a transaction that is suggested by the workflows to be monitored, through the creation of commitments by the monitor, without having it monitored. This means that the external agent is free to behave as it wishes; for example, to buy something from another agent without assuming the commitment to pay for it. It is free to try to do it, and the seller is free to accept that or refuse and ask to do the transaction with the creation of a formal commitment. This brings to our model an approach of modelling the human society model of doing business. Who knows if a seller needs to reduce the constraints of a formal order to sell more and assume risks? Humans do that all the time and we believe that a model for the implementation of open multiagent systems should allow for that to happen. The seller might check the reputation of the agent before doing business with it through an unmonitored interaction, or not. It is its own responsibility. Evidence of that behaviour of humans can be found in the example of the large number of

buyers and sellers on TradeMe (www.trademe.co.nz) and eBay (www.ebay.com), who engage in market exchanges that often have lower costs but also have higher risks.

The institutional environment mechanisms to avoid bad behaviour are implemented and ready to be used, the same way there are mechanisms for humans to constrain transactions among them, but the External Agent has the ultimate choice and freedom to act in the way it prefers. In some cases less constrained environments have proved to be more attractive to humans, so they might be more attractive to agents that are models of humans. Totally constrained institutional environments can be built as well; it is the designer's choice.

Therefore, the Deputy Agent can be seen as a bridge for a conversation between External Agents through the transitions "Unmonitored Interaction Incoming" and "Unmonitored Interaction Outgoing".

The Reputation Agent can inform a Deputy Agent about a sanction applied to it by the Institution Agent due to reaching some predefined low level of reputation, or reward earned by it due to reaching some predefined high level of reputation score. These will depend on how the reputation mechanisms available in the institutional environment are used by the designer. Is not compulsory to implement sanctions and rewards; the reputation can be used only to refuse certain interactions from low reputation agents, for example.

The Monitor Agent and the Reputation Agent will supply messages with information that is necessary for the Deputy Agent to Accept or Refuse a commitment proposal. The Monitor Agent sends a commitment proposal message and the Reputation Agent sends an answer to a query for the reputation of the commitment's creditor. The Deputy Agent will request that the External Agent analyse the information that the institutional environment offers to decide if it will accept or refuse the commitment. The External Agent will then reply to the Monitor Agent accepting or refusing the commitment.

3.8. The Normative Space

The Institutional Environment is composed of a set of agents with roles well defined in it. Each role is modelled as a CPN that represents the workflow of an agent in the environment, and that CPN models all the possible dialogs that the agent can take part in the institutional environment, obviously using the Institutional Acts identified at design time.

The Normative Space is actually the institutional environment workflow, which is formed by the joining of all the smaller workflows that compose the system. The workflows communicate with each other through FIPA ACL Messages that are delivered in the In Places and retrieved from the Out Places of the smaller workflows. These messages navigate in a FIPA ACL Message Exchange layer, which guarantees delivery of FIPA ACL messages to the agents registered with the institutional environment.

Therefore, the Normative Space, depicted in Figure 3.8, can be seen as a representation of the FIPA ACL based message delivery connections that represent a workflow that is composed by all the conversations that take place among the agents in the institutional environment. The CPNs that represent the roles of the systems agents and deputy agents group together and merge in the In and Out places forming a whole CPN that comprehends all the possible conversations that can take place in the institutional environment using the identified institutional acts. It is important to mention that in Figure 3.8 the In places represent all the Incoming Message places from the workflow CPNs and, similarly, the Out places represent all the Outgoing Message places present in the workflows.

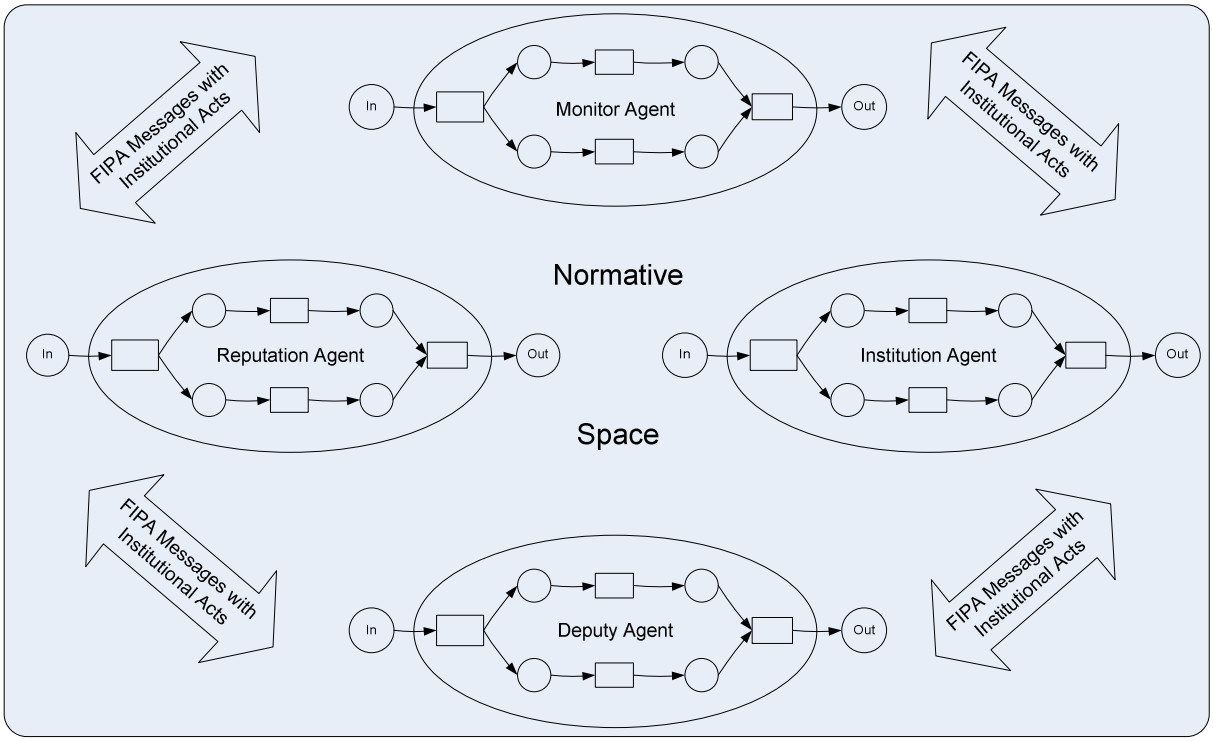


Figure 3.8: Normative Space of the Institutional Environment

We define it as Normative Space because we model a normative system in which all the identified norms that regulate the institutional environment are modelled in the

conversations that require the creation and management of commitment objects. These commitment objects are responsible for updating the reputation system implemented in the institutional environment and serve as a way to induce the agents to comply with the norms, and consequently build a good reputation score with the possibility of being awarded rewards; or break the rules and deal with the consequences that are mainly a lack of interactivity in the institutional environment or some more elaborate sanctions.

3.10. Freedom of Speech

Conversations that are not constructed from the predefined institutional acts can take place in the institutional environment among the Deputy Agents. These will be unmonitored by the Monitor Agent and represent the freedom of speech of the agents inside the institutional environment, which would be the meeting place for the agents.

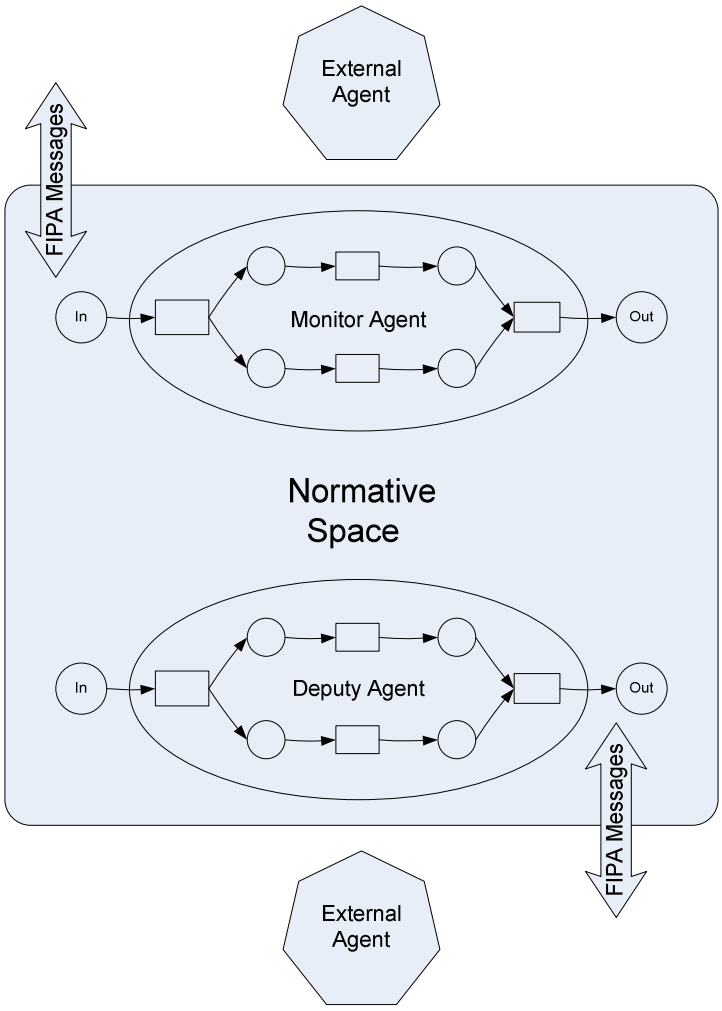


Figure 3.9: Freedom of Speech

The unmonitored interactions are used by the external agents at their own risk; for some reason an agent might want to engage in an interaction without the constraint of the creation of a commitment, and they are free to do it. Of course there are certain rules that the external agents cannot escape from; these are the social norms that it accepts when accepting to be debtor of commitments with the system agents, mainly the institution agent. But an external agent might know another external agent from before and decide to lower the restrictions applied by a monitored interaction and engage in conversations with that agent.

The freedom of speech is the way we found to introduce in our model an approach more adapted to the human model of engaging in conversations. Sometimes before doing business with somebody, we call the responsible authorities and check the person we are dealing with, but very often due to a lack of necessity and maybe even intentionally we do not check anything and just deal with the consequences. We believe that this characteristic should be present in a model that implements open multi-agent systems, because autonomous agents have their own objectives. Therefore, if the open system should allow for the joining of agents of any kind, of course it would depend on the restrictions the designer wants to impose in the system.

For example, a marketing strategy of an external agent might be to explore the greatest variety of profiles from the external agents. And some people just do not like to sign papers and assume commitments. That marketing strategy can be applied in our model because the external agent will decide if it only accepts certain transactions over monitored mode or in some cases it will allow unmonitored transactions due to certain characteristics of the external agent it is dealing with.

Furthermore, in a similar way external agents might check the reputation of an agent prior to engaging in certain interactions, and decide not to do it, or check the reputation but decide their actions based on their own history.

Freedom of speech might give an idea that our model allows for the lack of norms, but this is not true. Actually the social norms, represented by the commitments that the external agents assume with the system agents exist, and are compulsorily assumed by the external agents when they assume a role in the institutional environment. This means that the external agents have a set of social commitments pre-set for the external roles that they might assume in the institutional environment. Furthermore, all the Institutional Acts that

need commitments to be executed have the monitored interactions facilities attached to it. However the external agent has the ultimate choice to use unmonitored interactions nonetheless.

The CPN workflows model the conversations that take place in the institutional environment. Each System Agent and each Deputy Agent have a workflow modelled based on the role they play in the institutional environment, as described at the beginning of this chapter.

For every role that can be played by an external agent in the institutional environment there is a workflow modelled. These workflows are modelled based on the institutional acts that the external agents can use in the institution. Observe that this is to help the external agents designers to model or predict what is going to happen in the institutional environment.

The complete merge of CPNs forms the normative space of the institutional environment. Other than unmonitored interactions, all the conversations are predicted through the definition of the Institutional Acts that can be used in the Institutional Environment. The external agents can meet in the institutional environment or use the facilities made available in the Institution Agent by the Opal Agent Platform (Nowostawski, Purvis and Cranefield, 2002) to find each other, or to find specific agents that play specific roles in the institutional environment (because of their own goals) and carry out whatever conversation they want. It is important to understand that this scenario is beyond the conversations modelled by the institutional environment architect. The task is left to the developers of the external agents to deal with conversations that are not anticipated by the institutional agent modeller when developing the normative space of the institution.

This is what we call the freedom of speech allowed to the external agents. They have rules to obey but they have as well their own goals and even though these goals might be to disrupt the good functioning of the institutional environment, the institutional environment must implement mechanisms to deal with the misbehaving agents. The norms are there to be obeyed by the external agents. If they disobey, sanctions will be employed, in some cases more severely, for example expelling the unwanted agent from the institutional environment or, less severely, by allowing the agent to stay in the institutional environment but with a bad reputation. This will drive the agents out of important dialogs that take place as monitored interactions.

Freedom of speech is an important characteristic of a true open system that allows the external agents to pursue their own goals. This is as it occurs with human agents in society where people always pursue their personal goals or sometimes subsume their plans because of collective goals or rules of the society. We believe that with this feature our approach is more inclusive therefore our framework is more likely to be practically useful.

3.11. Institutional Environment State

The normative space is constantly updated by the exchange of messages between the external agents and the system agents through the deputy agents. The state of the institutional environment can be visualized in three ways:

- Snapshots of the CPN that represent the workflow of a specific agent shows the state of that agent;
- Snapshots of the CPN that represents the workflow of the whole Institutional Environment show the state of the Institutional Environment in relation to the exchange of messages that is happening at some point in time;
- The state of the commitments that are managed by the Monitor Agent;

The model that is updated by the Model Checker, located in the Institution Agent, gives a notion of state for the propositions that will be taken into consideration when evaluating the formulas that represent commitments, conditions and content.

The Model Checker is updated by the addition of states containing facts that are asserted in the institutional environment in some point in time. These facts change the truth value of propositions that wake up states of a model that represents the truth value of the propositions at some point in time.

Temporal propositions, defined as formulas in the Model Checker, represent the conditions and content of commitments. The evaluation of those formulas will verify if at some point in time they hold. If the state present in the model of the Model Checker can be reached and if it is composed by propositions that make one formula hold, the commitment will have its state updated to *Active*, *Fulfilled* or *Violated*.

Therefore, we can say that there are three mechanisms that inform the state of the institutional environment: the snapshot of the normative space, which tells what is happening in relation to the conversations that take part in the institutional environment; the current state of the model present in the Model Checker, which tells what is happening in relation to the commitment's conditions that set the state of commitments to *active* and the commitment's contents and the states of the commitment objects. This last mechanism tells what compromises are assumed by the agents in the institutional environment and in what state they are at some point in time.

3.12. Engineering an Institutional Environment

For the development of an Institutional Environment the system agents are provided together with the generic deputy agent, which embeds all the interaction protocols related with the services offered by the Institutional Environment.

When defining the roles that will represent external agents in the Institutional Environment, the Institutional Environment Designer will develop the subnets (see double rectangles in Figures 3.6 and 3.7) that consider specific institutional acts related with the application domain in which the Institutional Environment is built.

The definition of the social commitments and regular commitments should also be done at design time. The social commitments are those that the external agents have with the Institutional Environment that represent the norms of the system. The regular commitments are specific between roles defined in the institutional environment, reflecting audited interactions on request of the external agents.

Count-as rules should be defined as well. For the WikiCrimes Institutional Environment prototype described in Chapters 5 and 6, we have used events from the system to check situations where the low level concerns should be translated into institutional facts that are asserted in states inserted in the model present is the Institutional Agent Model Checker.

The system for reputation calculation should be pre-defined at design time as well and the policy for rewarding or punishing agents based on the reputation points the external agents have in the Institutional Environment. This relates to social control and the agents with lower reputation might be less often contacted for certain kinds of interactions depending on the objectives of the external agents. The reward and punishment system is based on

resources available in the institutional environment including their membership in the system.

3.13. Summary

In this chapter Institutional Environments were defined and all their elements were specified. The conceptual framework was defined from Section 3.1 to Section 3.6. Details on how the elements introduced in these sections are implemented as CPNs were given in Section 3.7. The high level architecture of the Institutional Environment was shown in Section 3.8 and the reusable elements to be used when making an Institutional Environment were highlighted in Section 3.12. The next chapter introduces all the technology employed to implement a framework based on the specification made in this chapter.

Chapter 4

The Technology Employed

This chapter presents a brief overview of the essential tools used in this thesis work and how they were employed to implement institutional environments.

4.1. Implementational Approach

To implement the Institutional Environments Model we employed a set of technologies available open source, some being developed by the New Zealand Distributed Information Systems Research Group from the University of Otago.

The main module was developed using the language Java from the object-oriented paradigm. All the other modules and tools communicate with the main module developed in Java. This choice was made due to the well-reasoned and stable version of the Java Language and that most of the other tools used are being developed in this programming language, thus facilitating integration.

The first tool chosen to use was the Otago Agent Platform (Opal) (Nowostawski, Purvis and Cranefield, 2002). This platform was developed in Java and is based on the FIPA Abstract Architecture and offers all the mechanisms for the management of FIPA ACL messages and implementation of software agents capable of handling these Messages.

The second tool chosen to use was the JFern CPN Tool Box (Nowostawski, 2009). This tool offers all the mechanisms for creation and execution of Coloured Petri Nets, the formalism chosen to model workflows in the institutional environments defined in this thesis. The tool was also developed in Java so it was very suitable for integration with Opal.

JFern offers the possibility of using the functional language Clojure (Halloway, 2009) to define the CPN inscriptions. That, as well, facilitated the work of manipulating FIPA ACL messages by the CPNs because Clojure runs on the Java Virtual Machine.

The third tool chosen was the Expectation Manager (Cranefield and Winikoff, 2009). This was developed in Python programming language and generated a bit more work to integrate with the main module developed in Java. The Expectation Manager has a model checker implemented that evaluates Linear Temporal Logic formulas. We use these formulas to define the condition and content of commitments that are evaluated against the model present in the model checker. The model represents a history of what is happening in the institutional environment. The solution for the integration of the model checker with our main module was the use of a Java bridge with the Python Language called Jython. Jython offers a Python interpreter that can be manipulated from within a Java source at run time. This interpreter is present in the Institution Agent and runs the model checker.

4.2. The Otago Agent Platform (Opal)

Opal (Nowostawski, Purvis and Cranefield, 2002) is implemented based on an architecture where agents are built from other agents. The so-called micro-agents are simpler components that do not have complex mechanisms usually associated with agents. They represent an abstraction where complex agents can be modelled and implemented based on them as their construction units. As objects can be comprised of other objects, complex agents could be composed of these simpler agents. Micro-agents can be used as building blocks for more complex architectures that encapsulate functionalities necessary for the management of an open system. Therefore, micro-agents can be used to define coarse-grained agents.

The Foundation for Intelligent Physical Agents (FIPA) has developed specifications to support the implementation of coarse-grained agent systems (FIPA, 2000). The FIPA Abstract Architecture (FAA) (FIPA, 2000) provides an abstract specification that can be used to develop agent platforms.

The Otago Agent Platform (Opal) was built based on micro-agents, and provides an instantiation of the FAA, together with other tools to support the development of agent based systems (Nowostawski, Purvis and Cranefield, 2002).

4.2.1. Opal Architecture

Opal is built in the form of an agent platform (AP). The key services that the AP provides are inter-platform message transport via the Message Transport System (MTS), agent management and a white-pages directory via the Agent Management System (AMS), and yellow-pages directory services via the Directory Facilitator (DF), as described in Nowostawski, Purvis and Cranefield (2002).

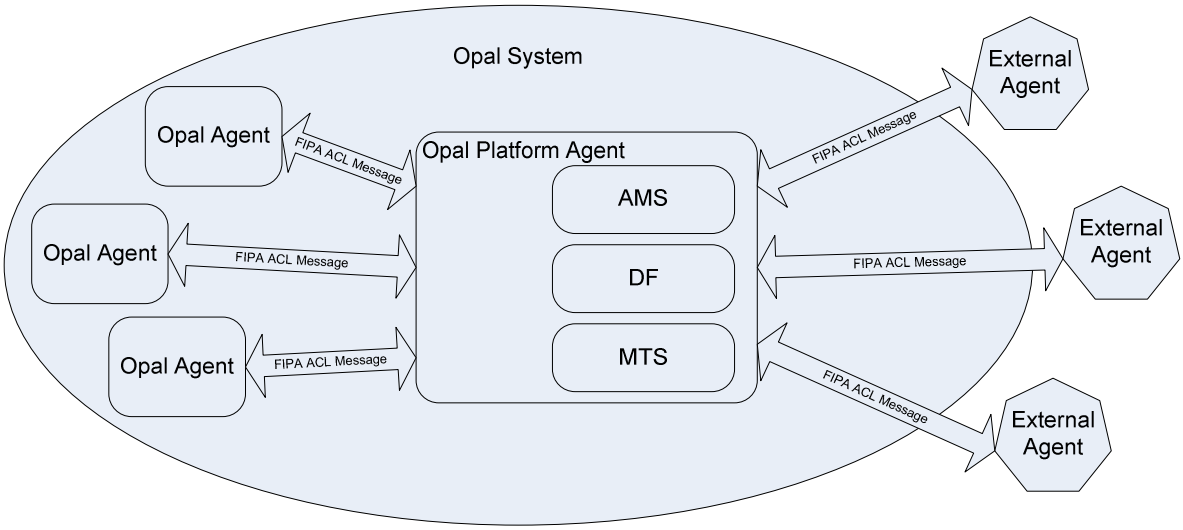


Figure 4.1. An Opal System implemented by Opal Agents.

A complete Opal Agent System with associated FIPA-specified services is depicted in Figure 4.1. Individual Opal Agents can all access an Opal Platform Agent. The Opal Management Console, which facilitates user monitoring and control of the Directory Facilitator, agent messages, and the AMS, is shown in Figures 4.2 and 4.3.

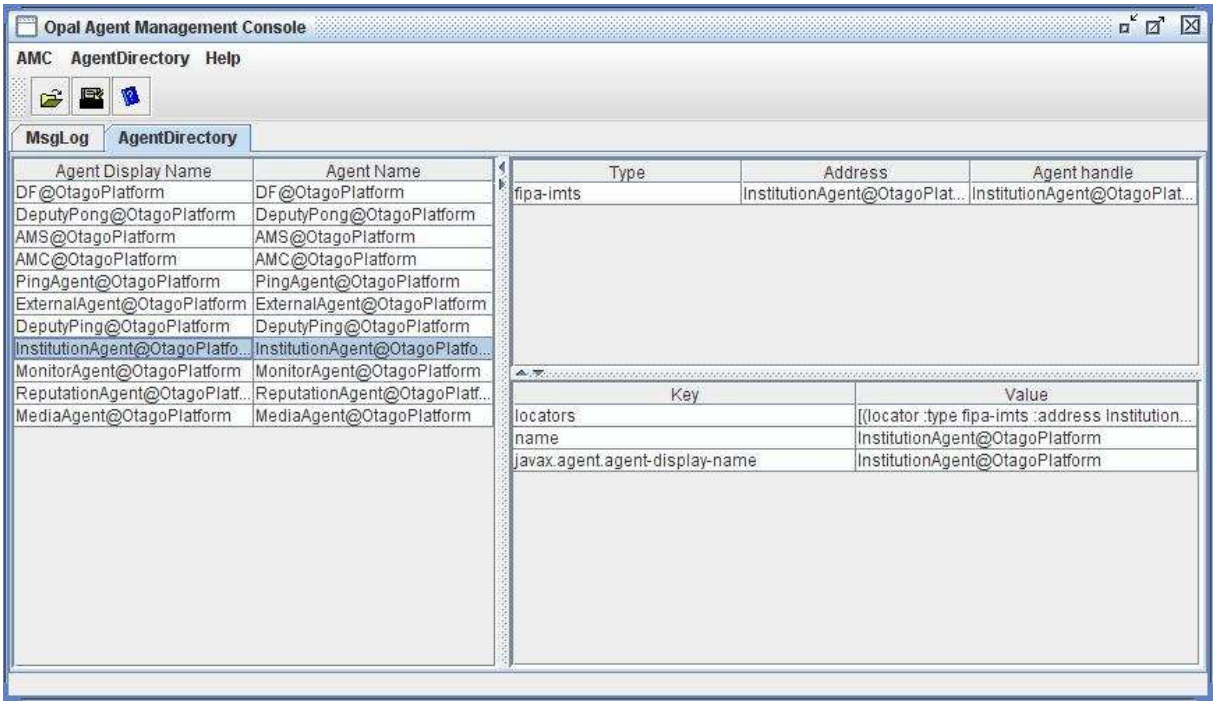


Figure 4.2. Opal Management Console

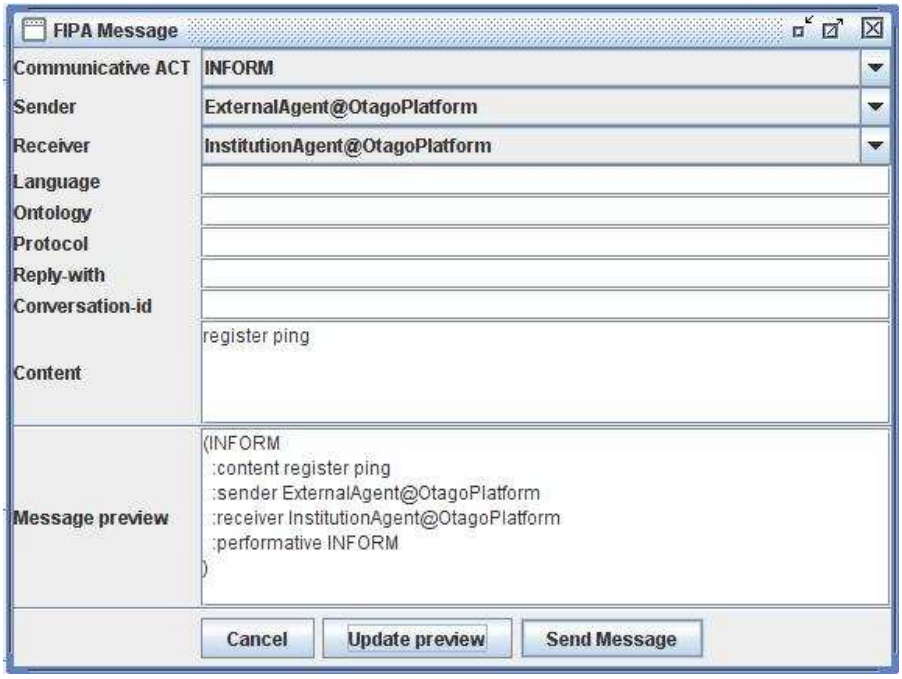


Figure 4.3. FIPA ACL Message being prepared in the Opal Agent Management Console

4.3. Interoperability with the Normative Space

There are two possible ways of interoperability between Opal and the workflows. First the messages exchanged among the workflows are carried by the message management

facilities implemented by Opal platform and, second, some transitions only consume tokens at the CPN level and execute actions that can be seen as side effects that interfere directly with the behaviour of the Opal Agents involved in the interaction.

The connection between workflows is made by the special In and Out places. Each workflow has a set of In places, one for each agent that can send a message to that workflow, but only one shared In place and one shared Out place for all the Deputy Agents that represent external agents. This is because the workflow predefines which agents can use which institutional actions directed to which other agents. All this is defined in the modelling of the institutional environment when identifying the possible institutional acts that can be used. When this is done it is also defined which agents can use each institutional act and to which agent it can address it. Therefore, the In places will receive messages from specific roles and the Out Places will receive messages that should be sent to other roles. Those roles belong to a particular institutional environment and are identified in the application domain. For each role identified, a CPN is modelled and implemented representing the workflow of that agent. This combines with the other workflows in the In Places and Out Places to form the normative space of the Institutional Environment, see Figure 4.4.

The Opal platform implements a FIPA ACL message exchange layer used in our implementation to guarantee message delivery between agents in the institutional environment. There is a fine integration between the Opal platform and the JFern CPN Tool Box.

In our implementation the CPNs are modelled using the JFern CPN modelling tool and they run in the JFern simulator. Therefore, there was the need to integrate JFern and Opal so that the agents can run their CPNs and send and receive FIPA ACL messages.

In Opal you can extend an OpalAgent class. This class has all the message exchange facilities accessible. Each OpalAgent has a number of behaviours attached to it and these behaviours implement specifics of what the OpalAgent does when it receives messages, including sending messages in reply to other OpalAgents. In our implementation we use the behaviours to connect the OpalAgent to the JFern simulator that runs a workflow for the OpalAgent.

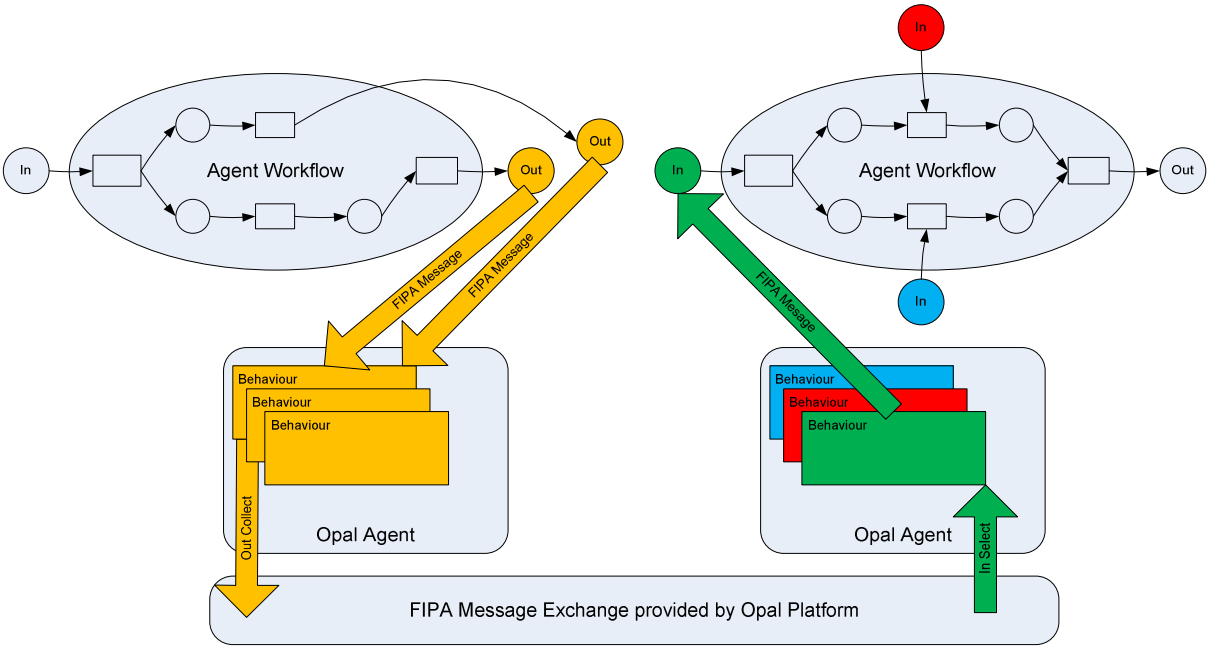


Figure 4.4. Message Exchange facilities provided by Opal

We implement a behaviour that is a message filter, for each In place of the workflow (represented by a naming convention). This means that the incoming messages from the Opal message layer will be filtered and put in an In Place modelled to receive messages from a specific role. These messages will be dealt with in the agent workflow, and after the processing of the workflow some messages will be put in the Out places of the workflow (also represented by a naming convention). These messages are collected by the OpalAgent and delivered to other OpalAgents through the Opal message exchange layer. With that we have an integration of FIPA ACL messaging exchange and CPN execution. In prior work, (Purvis, Nowostawski and Cranefield, 2002) and (Purvis et al., 2003), the approach used was to have a transition at the CPN level which would be responsible for the handling of the messages that were arriving to the In places of the CPNs.

Another form of integration from the JFern CPN Tool Box and Opal is identified when tokens are consumed by a transition that will run some code directly connected to the OpalAgent behaviour, not producing output tokens. This is seen as a side effect of the CPN and is implemented by the transition actions that can be defined for each transition in a CPN. These actions could be the manipulation of a database or a data structure that is present at the Opal level and not present at the JFern level. This is basically for the implementation of low-level actions that manipulate low-level system objects and are not present at a more abstract level.

The necessary side effects are implemented in the structure of the System Agents and the generic workflow of the Deputy Agent. Therefore it is not necessary for the Institutional Environment Designer to worry about the side effects, and he will not need to deal with the low level structure of Opal Agents when modelling the workflows of Deputy Agents for any given application domain. The system agents are already implemented for them.

There are four levels of abstraction identified in the implementation: the agency level implemented by Opal; the conversation level implemented by the union of the agents' workflows represented as the normative space of the institutional environment and executed by the JFern CPN Tool; the commitments layer which is a service implemented by the institution agent and the monitor agent (the first with the implementation of the model checker, which uses Linear Temporal Logic to take care of commitment conditions and contents, and the second with the management of commitment objects); and the normative layer which is implemented by the monitor agent and the reputation agent through the update of external agents' reputation due to the change of commitments' state.

It is important to notice that the agents have their conversations modelled through their workflows, which represent all the predicted dialogs that can take place using institutional action in the institutional environment. But there are the unmonitored interactions where the agents in the institutional environment can exchange their messages without the strict rules of the environment. This allows for the freedom of speech of the agents that will have the right to follow their own internal states. Of course, the conversations that are required to be monitored at design time are modelled as such in the workflows. But if the agents can understand each other they can exchange messages using other acts different from the institutional acts identified at design time, if they use the institutional acts they will be subject to the protocols modelled by the institutional environment and they will compulsorily participate in monitored conversations. When they do that they can break the rules and violate commitments but their reputation will reflect their actions in the environment and that can be expensive over time, depending on how the sanctions are implemented by the institutional environment architect.

The initial states of a commitment are set through the initial messages exchanged between the creditor and debtor of a commitment and the monitor agent. Initially the creditor asks the monitor to create a commitment, the monitor then proposes the commitment to the debtor, which can accept or refuse the commitment, setting its state to *pending* or *cancelled*

respectively. If the state is set to *pending* the commitment is created and its condition and content are sent to the interpreter in the institution agent. Upon the verification of the temporal propositions that represent them the institution agent can request the monitor agent to change the state of the commitment to *cancelled* due to timeout, *active*, *fulfilled* or *violated*.

4.4. JFern Net Tool Box

4.4.1. Java Integration

The Java Fernlet (JFern) engine (Nowostawski, 2009) is a Petri Net tool box fully integrated with the Java programming language. In Jfern all its components can be programmed through java syntax without the use of any Guided User Interface (GUI), which makes of it a good tool to be used in a Java programming environment.

4.4.2. Net Structure

The network model is based on hierarchical coloured Petri Nets (Jensen, 1992) (Reisig, 1985). Due to some terminologies variation, Jfern is different from other tools like DesignCPN (Jensen, 1992) and Renew (see http://www.renew.de), which are completely GUI-based.

One of identified difference from the formal model of CPN is the abstraction of object-based tokens. In JFern the tokens are generic objects (java.lang.Object), meaning that places are colourless that contain generic objects, which could be of any abstract type.

4.4.3. Multiset

The multiset is a bag of tokens represented by a java.util.Collection interface, which is implemented by the org.rakiura.cpn.Multiset class.

4.4.4. Place

The place is a container of tokens and it is represented by the class org.rakiura.cpn.Place. They have OutputArcs and InputArcs in relation to the transitions to which they are associated. The place objects can be accessed using its name or id.

4.4.5. Transition

The transition can be enabled or disabled. The Petri net simulator can fire a transition that is enabled.When this occurs, tokens are taken out of the input place according with the InputArc expressions, and tokens are put in the output place according with the OutputArc expressions. As a side effect of the transition an action associated to the transition can be executed.The scope of named variables covers the transition, its input arcs and its output arcs. A transition guard represents a Boolean function that decides whether the transition is enabled.

4.4.6. Input Arc

Input arcs connects input places to transitions and are represented by the class org.rakiura.cpn.InputArc. They have an expression associated that is used to select tokens to be consumed by the transition. They have as well input arc guard that evaluates to true or false, which notifies the simulator if a biding for the arc is valid or not. It is possible to use different languages supported to write guards and expressions, in our case we have used Clojure. Input arc guards are not a standard feature of CPNs but they have been added to JFern for convenience.

4.4.7. Output Arc

Represented by the class org.rakiura.cpn.OutputArc, these generate output tokens that are placed in output places. The output arc expression generates a multiset that might be empty or with some tokens that is associated with the output place. Tokens generated can be based on some that came from the input places or newly generated.

4.4.8. Building Workflows

JFern is a Java-based tool that makes available an editor to model CPNs graphically and runs the CPNs in simulators for testing. It is possible to add tokens in the places of the CPN or take tokens out of the CPN, run steps or the whole net until some final state is reached. It is a very good tool to model CPNs and simulate them before using it. JFern also makes available a Java API accessible from Java source code where the programmer can manipulate CPNs and run them directly from the Java source code at run time.

We have used the JFern Editor to model the agent roles, which we call agent workflows, and have accessed the CPNs directly from Java source code when coding the behaviours of the OpalAgents in Opal.

Clojure was the inscription language used in JFern in our implementation. Clojure is a Lisp dialect that runs on the Java Virtual Machine; it offers a great integration with Java and allows for the coding of the guards and arc expressions with much less code than Java itself. Using Clojure we can manipulate all kinds of Java objects and even instantiate them directly offering a solution more efficient and elegant for the coding of the workflows.

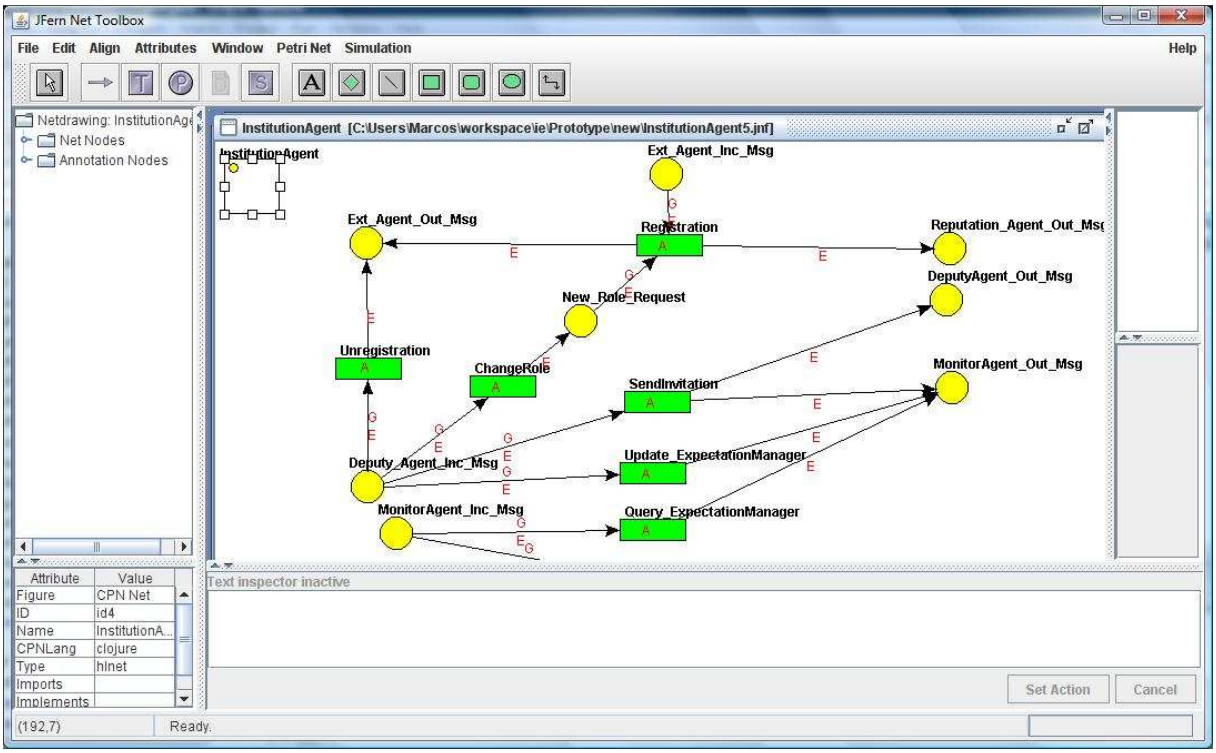


Figure 4.5. JFern Editor

4.4.9. CPN Examples

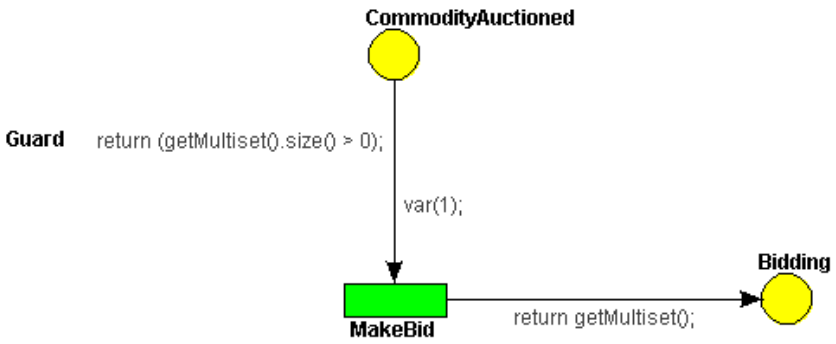


Figure 4.6. Simple auction example with Java Inscriptions

Figure 4.6 shows a snapshot of a simple CPN designed using JFern (Nowostawski, 2009). This defines a very simple auction where commodities are selected as soon as they arrive in the input place CommodityAuctioned. The transition MakeBid does some processing with the token and the bid is put in the output place named Bidding. The annotations in the CPN from Figure 4.6 are in the Java language, which is the default language for annotations in JFern. The guard is evaluated and returns true or false. If the guard returns true the arc expression (var(1)) is evaluated and returns the tokens that arrive in the place CommodityAuctioned as soon as they arrive. The arc expression from the input arc sets an unnamed variable to store the token selected by the guard. The arc expression from the output arc returns tokens from a multiset created by the selection made by the arc expression from the input arc and adds them to the Bidding place. The selected tokens can be used in the transition action code.

In Figure 4.7 subnets are represented as double rectangles that specify a link from a higher level net to a lower level net. Tokens are copied as soon as they arrive at the input places to the subnet, and they are processed in this net until the addition of tokens in the output places of the subnet. In Figure 4.7, “G” stands for Guard, “E” stands for arc expression, and these letters can be clicked on in the JFern editor to show the associated expression. Actions can also be associated with transitions and are indicated by a clickable letter “A” on the transition.

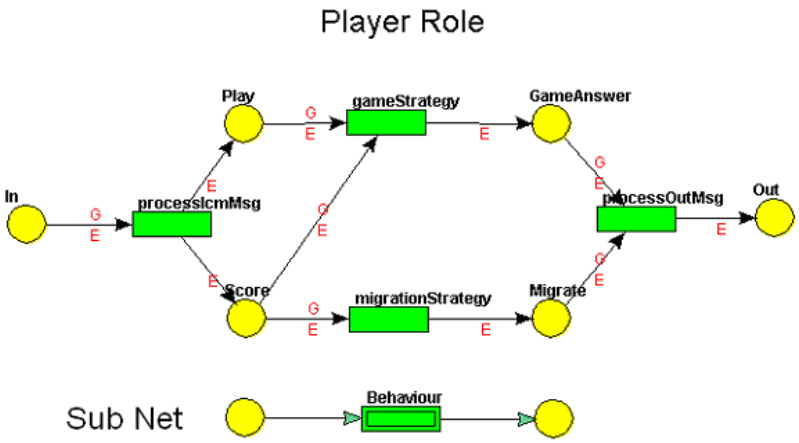


Figure 4.7. CPN Representing a player role and graphical representation of a subnet

To construct the institutional environment workflow, the various CPNs that implement the roles of agents participanting in the environment are implemented as subnets of the workflow. The lower level nets connect to form a higher level net that represents the workflow of the system. Theoretical accounts and practical implementation of hierarchcal CPNs have been addressed in prior research (Nowostawski, 2009) (Jensen, 1992).

4.5. The Model Checker

The Expectation Manager (Cranefield and Winikoff, 2011) running in the Institution Agent makes available a model checker that contains the state of the Institutional Environment regarding propositions that represent facts that are asserted about the state of affairs defined in the condition and content of commitment objects. The model is updated at run time and new states are inserted into it when new facts are asserted according to the conversations that take place in the institutional environment.

The Expectation Manager is written in Python and we use a bridge between Python and Java, called Jython (Juneau et al., 2010), to integrate it with the Institution Agent that was developed as an Opal Agent in Java.

The Expectation Manager implements a Model Checker that evaluates formulas that represent temporal proposition in LTL logic. The temporal logic is used to represent the temporal propositions present in the condition and content of commitments. This is necessary due to the temporal characteristics of these two commitment's attributes.

The commitments are objects that have a well-defined life cycle. These objects are managed by the Monitor Agent and have their state updated through messages coming from the Institutional Agent that has the model checker for the verification of the condition and content of the commitment objects.

Count-as rules are used to separate implementational concerns from the more abstract concepts found in the application domain. The count-as rules are implemented in the behaviour of the Opal Agents and translate into facts present in states that are inserted in the expectation manager. They observe low-level facts, such as, for example, institutional agent is online, and generate propositions with a truth value that are part of a state that is inserted at run time in the model checker. These propositions can be seen as facts that are used to evaluate formulas that represent the condition and content of commitments changing their state to *active*, *fulfilled* or *violated*, for example.

In the example Counts-As rule below we can see that *Report_has_status_accepted* is a proposition that influences in a higher level of abstraction and is evaluated to true or false based in lower-level facts expressed in the counts-as rule. The lower-level facts are evaluated in the behaviour of the Opal Agents, implemented in their MessageFilter class, and the more abstract proposition will be part of a state that is inserted in the history of the model checker.

InstitutionAgent_is_Online
and *MonitorAgent_is_Online*
and *ReputationAgent_is_Online*
and *DatabaseAdministratorAgent_is_Online*
and *DeputyRegisteredUser_is_Online*
and *DatabaseAdministratorAgent_receives_RegisterCriminalFactMessage*
counts as *Report_has_status_accepted*

The model checker itself is not necessary to evaluate rules, but instead we have an evaluation of formulas in the model checker whenever a new state is inserted in it being the output of the evaluation sent to the Monitor Agent as FIPA ACL Messages that update the state of the commitments managed by it. The model checker is used to evaluate temporal

propositions, therefore we do not use some special purpose operators of the Expectation Manager, such as ExistsExp, ExistsFulf and ExistsViol (Cranefield and Winikoff, 2011). What is done is only the check of value of formulae involving propositions, conjunctions, negations and standard temporal operators such as Next and Until.

4.6. Summary

This chapter presented a brief overview of the essential tools used in this thesis work and how they were employed to implement institutional environments. A better view in what customizations are necessary to make an Institutional Environment can be found in chapters 5 and 6 where there is the definition of the WikiCrimes domain and execution of scenarios in that domain respectively.

Chapter 5

The WikiCrimes Institutional Environment

This chapter introduces the case study of this thesis. The WikiCrimes System and its suitability to be implemented as an open MAS are presented and the WikiCrimes Institutional Environment is specified.

5.1. Introduction

The explanation about the WikiCrimes collaborative system (www.wikicrimes.org) is copied from the article Furtado et al. (2010). The general public is making ever greater use of collaborative systems. This momentum of collaboration is mainly leveraged by the Web 2.0 (O'Reilly, 2007), in which the difference between information producers and consumers decreases significantly, since several applications emphasize the production of information by any Internet user. A culture of sharing induces people to externalize their feelings, opinions, experiences and even their goods. With the mass production of content, collaboration mechanisms such as Wikipedia (http://www.wikipedia.org) came almost naturally.

Despite the huge success of Wikipedia, finding good and useful causes that are capable of involving thousands or even millions of people operating with little or no coordination is still a big challenge. A slight variation of the wiki concept illustrated by Wikipedia has been dubbed crowdsourcing (Howe, 2008). In that context, mass collaboration occurs via human resource outsourcing based on awards (often financial) or even punishments that ‘‘motivate” people’s participation in large campaigns.

In the context of public and government areas – where social problems would be a potential target to be treated by means of mass collaboration – crowdsourcing is particularly difficult, since few investments are available to leverage people's participation.

In the law enforcement domain, besides the aforementioned difficulties, another particular challenge must be faced. In that context, finding the equilibrium between people's participation and information credibility is crucial. Anonymous mass collaboration is the easiest way to receive information; however, the credibility of the received information is depreciated, because the source of information is unknown.

It is in that particularly complex domain that the Knowledge Engineering Research Lab of the Informatics Department of the University of Fortaleza (www.unifor.br), has initiated a project called WikiCrimes (http://www.wikicrimes.org) (Furtado et al., 2010) The project is driven by three goals: (i) to give more transparency and publicity to criminal information, (ii) to provide means for citizen crime prevention, and (iii) to reduce the phenomenon of crime under-reporting (crimes that are not notified to authorities). These goals have been on the political agenda of several countries around the world, particularly those in which the population suffers from high rates of violence.

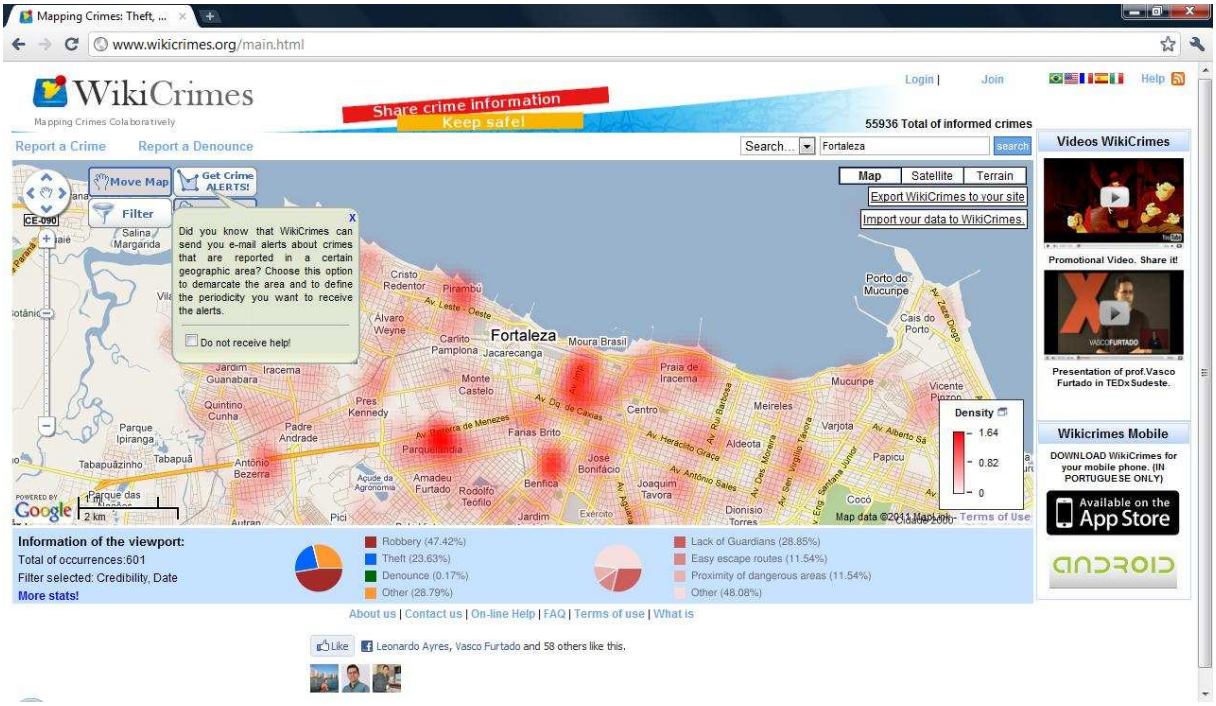


Figure 5.1: The WikiCrimes Collaborative System

WikiCrimes aims to offer a common interaction space among the public in general, so that they are able to notify other about criminal acts as well as keep track of the locations of those acts. The goal is to obtain collaborative individual participation for generating useful information for everyone. It is based on the principle that the ones who hold information about crimes are the citizens. If they want to make such information public, they can. Here, Furtado et al. (2010) are appealing to the feeling of sharing that is so usual in victims of violence. When someone is a victim of any type of crime, it is usual to tell someone else about the criminal act. Typically, those who are close to the victim are the first to be informed. What WikiCrimes intends is to provide an environment to be that ‘‘global blackboard” of stories told by people about crimes, to help alert other people on a scale larger than their closest social contacts. In other words, if there is participation, crime mapping starts to be done collaboratively, and everyone will benefit from having access to information about where crimes occur.

5.2. Collaborative Systems

We understand collaborative systems as computer tools that support the interaction (direct or indirect) among a certain number of people with individual and/or collective goals. Research into collaborative systems is divided into several areas and assumes different yet interrelated terminologies such as Groupware, Computer-Mediated-Communication (CMC) and Computer-Support Cooperative Work and Learning (CSCW/CSCL) (Fouss and Chang, 2000). ‘‘Computer-Mediated Communication” can be seen as the vast area that encompasses computer-mediated communication research, including collaborative systems. ‘‘Groupware” refers to technology applied to computers and networks designed to enable work in groups. This technology can be used to communicate, cooperate, coordinate, troubleshoot, compete and/or negotiate. Examples of groupware technologies are email, newsgroups, wikis, video conferences and chats (Grudin and Poltrock, 1991). CSCW and CSCL refer to the areas of study that will examine the design, adoption and use of groupware tools in the context of working and learning.

All these kinds of collaborative systems have been strongly influenced by the recent trends of the Web. The Web can be seen in terms of two distinct phases, and one way to distinguish these phases is to look at the number of producers and consumers of information. In the traditional Web (or Web 1.0), there are few producers and many consumers. The main producers are specialized persons (web designers), companies (both

public and private), and traditional media. The typical Internet user only accesses the information made available by these producers. In Web 2.0, the difference between producer and consumer of information decreases significantly, since several applications emphasize the production of information by any Internet user. The recent wave of blogs, photologs, wikis, online communities, social networks, etc., is an example of this context. Another type of application that is beginning to emerge is aimed at the integration of information from different sources (‘‘mashups”).

5.3. WikiCrimes: Motivation and Goals

The veracity and accuracy of information about where crimes occur, as well as the information on the characterization of such crimes, has always been on the agenda of discussions about public safety in Brazil and in various other countries. Traditionally, this information is monopolized by law enforcement agencies and is, therefore, characterized as a highly centralized mechanism. This monopoly ultimately creates tension in the relationship between such agencies and society at large, because it is commonly opposed to the precept of disclosure and transparency of information required by a democratic regime.

Allied to this context are the crises that have characterized the daily routines of law enforcement agencies, as well as their limitations to providing a quality public service, thus tending to diminish citizens’ trust in those agencies. These factors encompass some of the reasons for the growing problem of under-reporting – the low rate of reporting crimes – that has occurred (Kahn, 2008). It has become common for one to hear someone who has been mugged say that they did not file a police report because they thought it would not bring about any effect. Polls conducted with crime victims in several Brazilian states show that under-reporting may, in densely-populated areas, reach 60% for certain types of crimes. The result of this can be disastrous in terms of formulation of public policies and particularly in the planning of police actions, since the official crime mapping may be reflecting a trend that is somewhat different than what is actually occurring in real life.

The idea behind WikiCrimes is to provide a common area of interaction among people so that they can report and monitor the locations where crimes are occurring (Furtado et al., 2010). It is based on the principle that the ones who hold information about crimes are the citizens. If they want to make such information public, they can. Thus, individual participation, in a collaborative manner, can generate collective intelligence. In other

words, if there is participation, crime mapping starts being done collaboratively, and everyone will benefit from having access to information about where crimes occur. WikiCrimes is a typical Web 2.0 application. It allows users to access and to register criminal events on the computer directly in a specific geographic location represented by a map. That is one of the phases of the crime mapping activity in which an analyst collects, maps, visualizes, and analyzes crime incident patterns.

5.4. WikiCrimes Institutional Environment

The open and participatory characteristics identified in WikiCrimes makes it susceptible to abuses or fraud attempts. It is important that as many people as possible collaborate with the system, contributing to the growth of its data records. But it is equally important that the information registered in the system be reliable, so that the system can become a trustworthy source of information. In saying this, our claim is that an open collaborative system must be viewed as a kind of open multi-agent system, where a number of human and/or artificial agents interoperate pursuing their individual or common goals (De Oliveira et al., 2008). These agents can contribute positively or negatively to the organization and goals of the system as a whole, and the agents are free to join and leave the system as they wish, as long as they obey certain rules that must be observed concerning the management of the system.

The management of interoperation among agents is a complex task, and robust techniques and methodologies for the development of reliable and open Multi-Agent Systems (MAS) have been studied in academia (Aldewereld et al., 2006), (Arcos et al., 2005), (Colombetti, Fornara and Verdicchio, 2002), (Grossi et al., 2006), (Minsky and Ungureanu, 2000), (De Oliveira and Purvis, 2008), (Ricci and Ominici, 2003), (Zambonelli, Jennings and Wooldridge, 2003). Such techniques are aimed at the modelling and implementation of features that give openness to those agents, allowing them to have the ultimate choice of obeying regulations or dealing with possible sanctions imposed by the MAS norms. After all, agents are autonomous entities and the biggest challenge is to have a coordination system where the agents can be free to decide what to do, but at the same time be encouraged or enticed to obey the regulations of the artificial society they are entering.

We visualize the WikiCrimes Collaborative System as implementing an application domain suitable for the open MAS model. Therefore, we have investigated in this research the

adaptation of the WikiCrimes Collaborative System to the context of an Institutional Environment and use it as a case study for the framework developed. We have implemented the WikiCrimes Institutional Environment and what follows is the adaptation of the WikiCrimes application domain to the context of an Institutional Environment, where we define the elements that compose the prototype system developed. Chapter 6 demonstrates the execution of scenarios in the prototype showing how the WikiCrimes Institutional Environment behaves in specific situations where issues from the open MAS model are identified and handled by features of our framework, thus demonstrating the suitability of it to model and implement open MAS.

5.4.1. Roles present in the WikiCrimes Institutional Environment

The System Agents are present in the institutional environment: Institutional Agent, Monitor Agent, Reputation Agent, and Database Administrator Agent. The Deputy Agents that represent roles in the WikiCrimes Institutional Environment played by the External Agents are defined below:

- Browser User: the user that browses the institutional environment basically seeks information in the system. The institution will try to make this kind of user assume the Registered User Role.

- Registered User: in this role the agent will be a typical user of a collaborative system. In the case of the WikiCrimes Institutional Environment, it will be able to register crimes, browse the environment, confirm crimes, denounce abuse, disconfirm crimes and invite other agents to confirm crimes. These invited agents can be a Registered User or external agents that are not registered yet in the institutional environment but can become an Invited User only to confirm the crime and decide if they want to become Registered Users.

- Invited User: this role is played by the agent that is indicated to confirm a crime. The indication is done by an agent playing the Registered User role. This agent is one step closer to playing the Registered User role rather than the Browser User.

- Certifier Entity: this is a special kind of role played by agents that own a very respected position in the WikiCrimes Institutional Environment. This agent has the reputation of a System Agent; the difference is that the Registered Users should not

denounce abuses or disconfirm crimes registered by this category of agent. If this situation arises it could be an indication that the Certifier Entity can have their position in the WikiCrimes Institutional Environment reviewed or possibly that they have made a mistake.

5.4.2. Institutional Acts

Apart from the generic institutional acts that are defined for any institutional environment implemented using our approach, there are the specific institutional acts that relate to the application domain of the WikiCrimes System (Furtado et al., 2010). Below we list the institutional acts that initiate interaction protocols in the agents' workflows, and the respective agents that participate in the interaction. The agents that use the institutional act are listed in brackets, where 'generic' means that all the external agents can use the act and 'specific' means that only the listed roles use the act:

- (generic) *Registration*: The Institutional Agent allocates a Deputy Agent to the External Agent that is registering with the Institutional Environment. The Deputy Agent represents a role played by external agents in the Institutional Environment. The Reputation Agent creates a record for the new agent in the reputation database. The information requested from the External Agent is a valid email address.

- (generic) *Deregistration*: The Institution Agent deallocates the Deputy Agent assigned to the External Agent that is unregistering with the Institutional Environment. The Monitor Agent is informed so it can update commitment states and the Reputation Agent is contacted to update the reputation of the External Agent.

- (specific, Invited User requests to be Registered User, Browser User requests to be Registered User, Registered User requests to be Certifier Entity) *Change Role*: The External Agent requests through his Deputy Agent to the Institution Agent to assume a new role in the system. The Monitor Agent updates commitment states and the Reputation Agent updates the reputation of the external agent. A new Deputy Agent representing the new role in the Institutional Environment is allocated to the external agent and the former Deputy Agent is deallocated.

- (specific, Registered User, Certifier Entity) *Register a Criminal Fact*: A Registered User Deputy Agents communicates a new criminal fact to the Database Administrator Agent. The Deputy has to inform the Database Administrator Agent of the crime details and at least one email of an external agent. The email is used by the Institutional Agent to invite the External Agent to confirm the crime positively or negatively. The Database Administrator Agent checks if the new criminal fact is in an observation area registered by some registered user; if this is the case it sends a message to the email of the registered user about the new criminal fact.

- (specific, Database Administrator Agent) *Invite to Confirm a Crime*: The Database Administrator Agent asks the Institutional Agent to check if the email address included in the registration of a criminal fact belongs to a Deputy Agent that is online in the Institutional Environment and Requests the Deputy to ask the External Agent that it represents to confirm the crime with the Institutional Agent. If there is no Deputy Agent online who owns the email address in question, the Institutional Agent sends an email to the External Agent asking him to come online as a Registered User and confirm the crime. If the email address is not registered with the Institutional Agent, it sends an email inviting the external agent to register in the Institutional Environment as an Invited User to confirm the crime and possibly change role to a Registered User.

- (specific, Monitor Agent) *Commitment Proposal*: The Monitor Agent requests to a Registered User Agent to accept to be a debtor in a pre-commitment (a commitment in the state *unset*). If the Registered User accepts, the pre-commitment becomes a commitment in the Institutional Environment.

- (specific, Registered User, Invited User) *Confirm a Crime*: The Registered User Agent or Invited User Agent sends a Message to the Database Administrator Agent confirming the crime positively or negatively. The negative confirmation is informed by the Database Administrator Agent to the Monitor Agent change the state from the commitment to tell the truth that the external agent has acquired when registering with the Institutional Environment. The reputation of the agent that is confirming the crime is checked by the Database Administrator Agent before the crime is confirmed. If the reputation is bad, the crime is not confirmed either positively or negatively. The email of the Invited User is registered in the

Reputation Database with the reputation agent and whenever the external agent changes his role to Registered User it will have that reputation assigned to it.

- (specific, Registered User) *Denouncing Abuse*: An external Agent with a Deputy representing a Registered User role communicates to the Database Administrator Agent that a specific crime registered has abusive information. The same treatment as for the confirmation of a crime is done here.

- (specific, Registered User) *Commenting on a Criminal Fact*: An external agent with Deputy Agent representing a Registered User communicates to the Database Administrator Agent a comment on a criminal fact. These comments could possibly be analysed to add or subtract trust in the reported crime[6].

- (specific, Registered User, Browser User, Invited User) *Commenting on the Institutional Environment*: The Registered User Agents inform the System Administrator Agent about a comment on the system.

- (specific, Registered User) *Registering an Observation Area*: A Registered User informs the Database Administrator Agent about an area that he wants to watch for new crimes.

- (specific, Registered User) *Deregistering an Observation Area*: A registered User informs the Database Administrator Agent about the deregistration of a registered observation area.

- (Generic) *Ask for a safe route*: A Registered User asks the Database Administrator Agent for a safe route between two locations.

- (Generic) *Query the Criminal Database*: a Browser User, Invited User or Registered User asks the Database Administrator Agent about crimes in specific areas.

- (Specific, Registered User) *Query the Reputation Database*: a Registered User asks the Reputation Agent about the reputation of another Registered User.

[6] This level of analyses is identified as a future work for this research and is out of the scope of this thesis.

- (Specific, Registered User) *Ask another Agent to be part of a group*: a Registered User asks another registered user to be part of a group of agents that register in a specific area. The agent checks the reputation of the other agents before asking.

5.4.3. WikiCrimes Institutional Environment Workflows

The workflows developed in the Institutional Environment are managed by the JFern CPN Tool Box (Nowostawski, 2009) (the annotations of the CPNs present in this chapter are shown in Appendix A). For each workflow, an instance of the tool is created for the visualization of the message passing among the agents that are registered with the institutional environment. Of course the number of visual instances of the tool can be managed to avoid overload of the system or visual confusion on the screen.

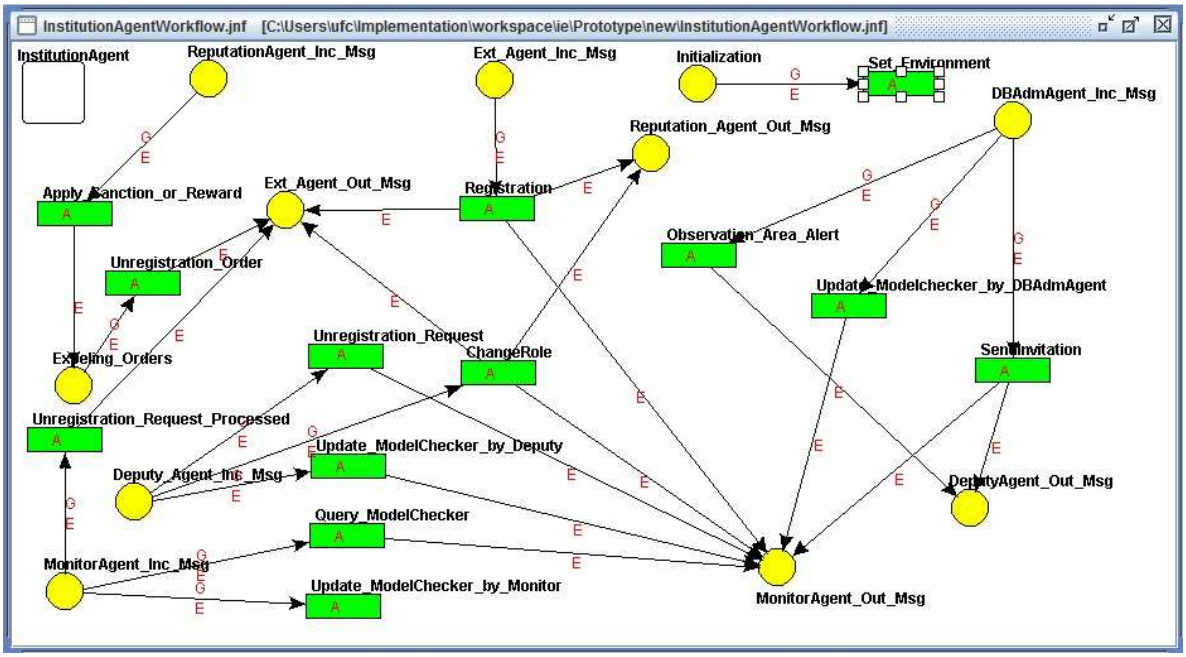


Figure 5.2. Institution Agent Workflow in JFern

Above, in Figure 5.2, the instance of the Institution Agent Workflow is visualized. This corresponds to Figure 3.3. The Figure depicts the behaviour of the Institution Agent when receiving the messages with the predefined set of Institutional Acts about the management of the Institutional Environment. Here it can be seen that there is a distinction between institutional acts that are related to the management of the institutional environment, and consequently considered generic and suited to any institutional environment; and the institutional acts more suited to the application domain for which the system was

developed. The latter will become more evident when defining the specific behaviour of the Deputy Agents that represent the External Agents in the institutional environment.

The Monitor Agent Workflow is depicted in Figure 5.3. This corresponds to Figure 3.4. It is designed to manage all the commitment objects instantiated in the institutional environment. The In and Out places communicate with the other workflows and some transitions only consume tokens, executing an action statement that represents a side effect of the transition without producing any output token.

The Agents that communicate with the Monitor agent are the Institution Agent and the Deputy Agents. They can communicate to propose commitments to other agents registered in the institutional environment or that might register in the future, in the case of an invitation to confirm a crime. The commitments proposed by the Institution Agent are considered social norms from the Institutional Environment.

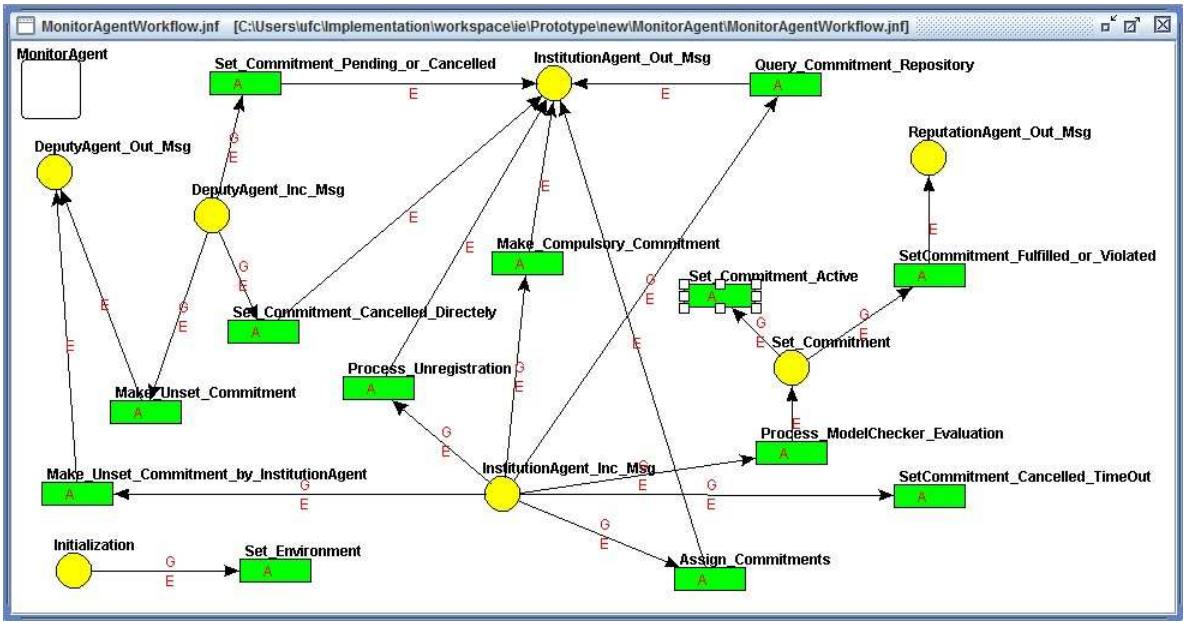


Figure 5.3. Monitor Agent Workflow in JFern

There is a high degree of communication between the Institution Agent and the Monitor Agent due to the condition and content of the commitments being represented as temporal propositions defined as formulas in the Model Checker present in the Institution Agent. These formulas are constantly evaluated by the Model Checker when states are inserted in the model defined in the Model Checker. Whenever the formulas evaluate to true, messages are sent from the Institution Agent to the Monitor Agent to update the commitment objects managed by it.

The Reputation Agent Workflow is depicted in Figure 5.4. This corresponds to Figure 3.5. It creates records for External Agents that are registering with the institutional environment; executes updates in the reputation of an agent due to requests from the Monitor Agent when commitments are evaluated to *fulfilled* or *violated*, and executes queries in its records about the reputation of a registered agent due to requests from the Deputy Agents.

The Database Administrator Agent is shown in Figure 5.5, which corresponds to Figure 3.6. This manages the Institutional Acts that are related to the storage or retrieval of information from a Database Management System. In the Case of the WikiCrimes Institutional Environment there are the general Institutional Acts, present to all the institutional environments that require this type of system agent.

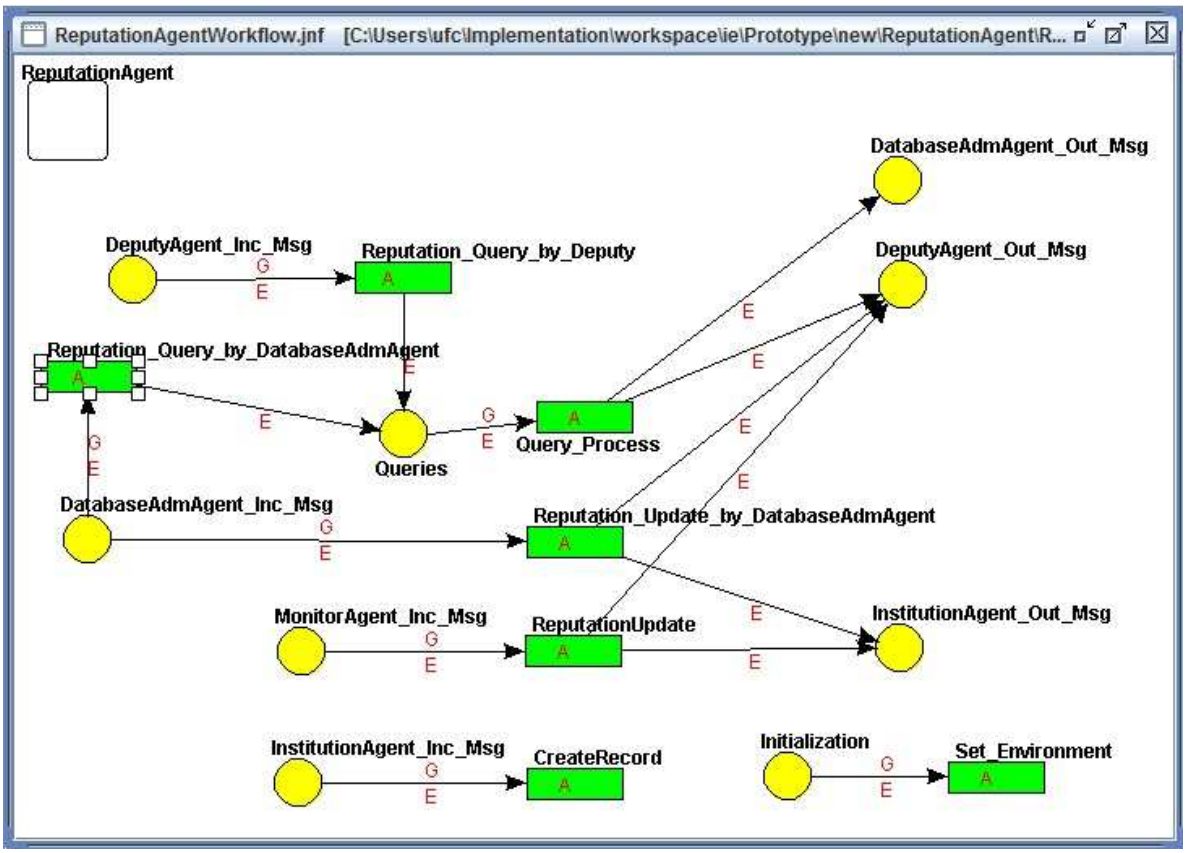


Figure 5.4. Reputation Agent Workflow in JFern

All the negotiation with the Database Management System, which manages the criminal database, is made by side effects implemented in the transitions of the Database Administrator Agent.

The Deputy Agents have a generic workflow, depicted in Figure 3.7. This workflow manages all the institutional acts that are used in any institutional environment. The

specification of the institutional acts that are specific to a role indentified in the institutional environment is done by the expansion of the subnet “Process specific institutional actions”. The subnet has all the input places connected to new transitions that form the particulars of the behaviour of the role and leads to output places drawn in the Deputy Agent Generic Workflow below.

The Deputy Registered User and the Deputy Certifier Entity have the same specific behaviour implemented, as shown in Figure 5.6. The distinction between these two roles is only in relation to the reputation model implemented in the WikiCrimes Institutional Environment. A Certifier Entity is a special kind of agent that represents a person or group of persons that have a high level of reputation that cannot be changed and should not be complained about. If this happens it might be an indication that the Certifier Entity has committed a mistake or that a Registered User is trying to change the image of this agent in the Institutional Environment. Examples of this kind of institution are the police department or a well-established newspaper. The Deputy Registered User does not have the same treatment in relation to its reputation in the WikiCrimes Institutional Environment. It has its reputation evaluated all the time by the observation of the commitments that it assumes in the institutional environment.

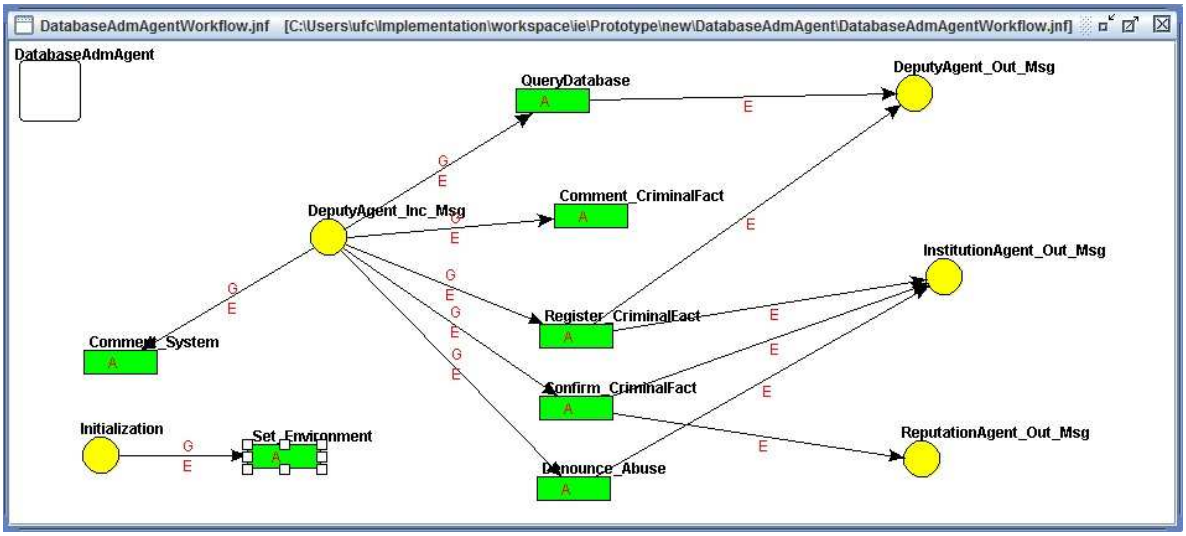


Figure 5.5. Database Administrator Agent in JFern

These two roles basically deal with the Database Management Agent to register a criminal fact, comment on a criminal fact, confirm a criminal fact and denounce abuses in the WikiCrimes Institutional Environment. They can also invite another deputy agent to be part of a group, for example, to join people from a neighbourhood or that have other particular

interests in registering criminal facts in that region or evaluating if the information registered in the criminal database can be confirmed positively or negatively.

The Registered User and Certifier Entity roles can as well invite other external agents to confirm a crime. This invitation is sent to the Institution Agent because it will check if the agent is already registered with the WikiCrimes Institutional Environment, or not. If the agent is registered it is proposed by the Institutional agent to assume the commitment of confirming a crime within a maximum of 15 days from the invitation, or, if the agent is not registered, an email is sent to the External Agent inviting it to join the institutional environment to confirm a crime. A pre-commitment is generated in this last case, and it becomes *active* if the external agent joins the WikiCrimes Institutional Environment and accepts the invitation to confirm the crime within a maximum of 15 days past the registration.

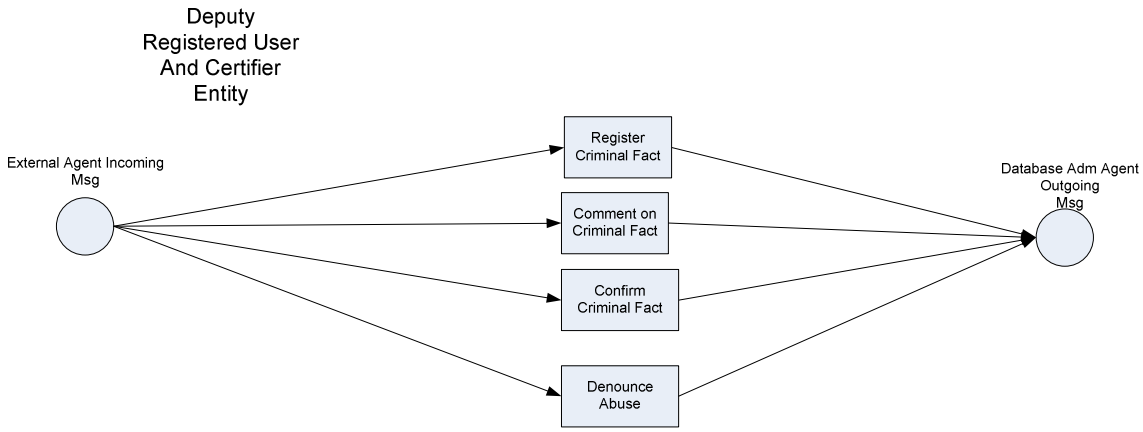


Figure 5.6. Specific Behaviour of the Deputy Registered User Agent and Deputy Certifier Entity Agent

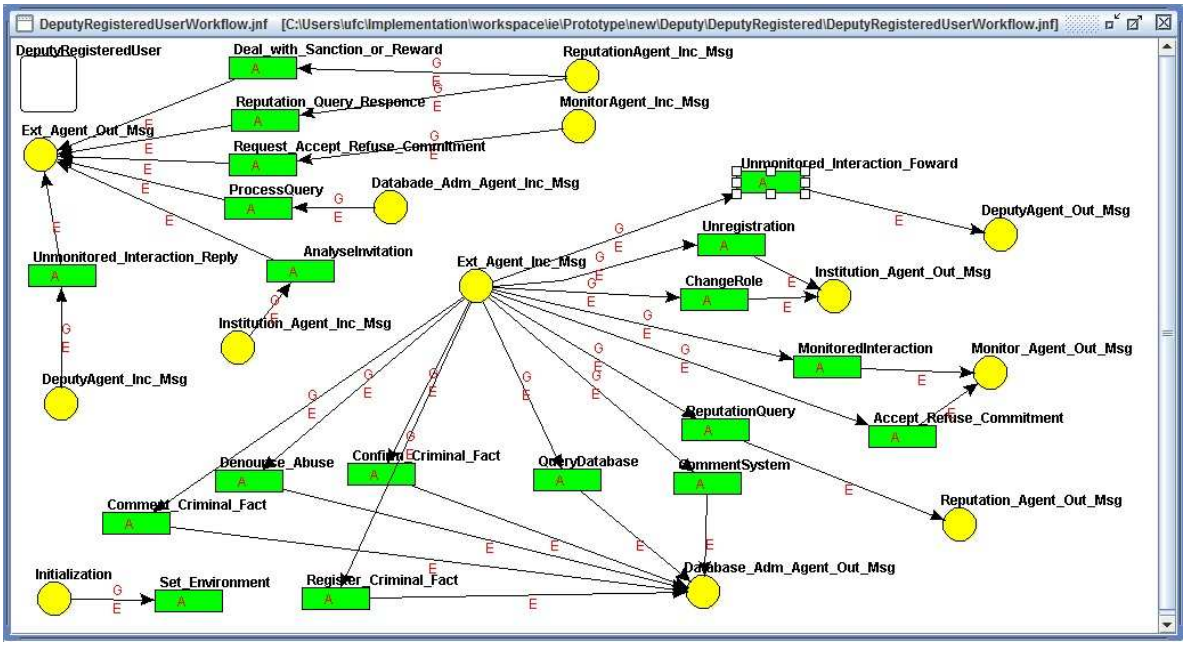


Figure 5.7. Deputy Registered User Workflow in JFern

The workflows of the Deputy Registered User and Deputy Certifier Entity were captured from the JFern CPN Tool Box and are depicted in Figures 5.7 and 5.8, respectively. Observe that the Deputy Agent Generic Workflow is present in the figures together with the specific behaviour of these two roles.

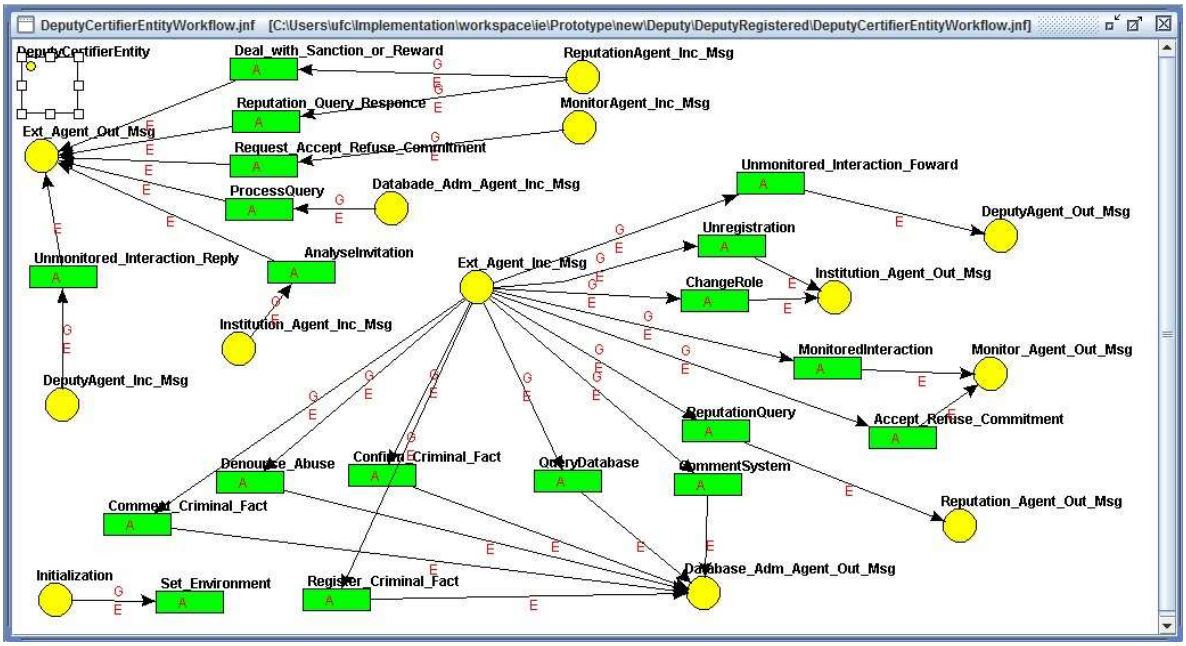


Figure 5.8. Deputy Certifier Entity Workflow in JFern

The particular behaviour of the Deputy Invited User role is represented in Figure 5.9. It shows that in this role an External Agent can confirm a criminal fact only.

Figure 5.10 shows the Invited User Workflow in the JFern CPN Tool Box as it was implemented. Again, the generic behaviour of any Deputy Agent is enhanced by the specific behaviour of this role.

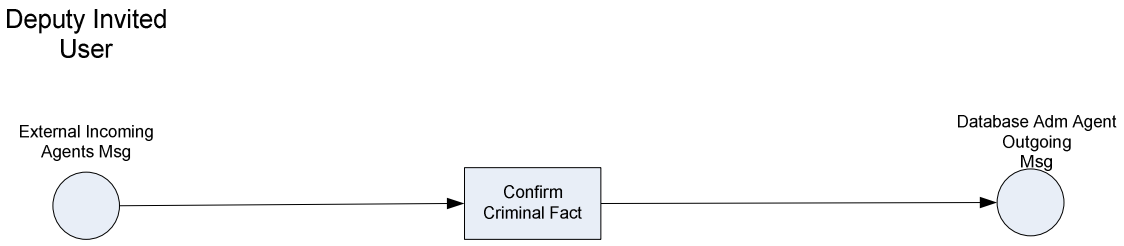


Figure 5.9. Specific Behaviour of the Deputy Invited User Agent

The Deputy Browser User Workflow represents a random user that is only browsing the criminal environment for information trying to meet other external agents that are registered with the WikiCrimes Institutional Environment. It is capable of using all the generic institutional acts defined in the institutional environment. Its workflow is shown in the Figure below, as it appears in the JFern CPN Tool Box where it was implemented.

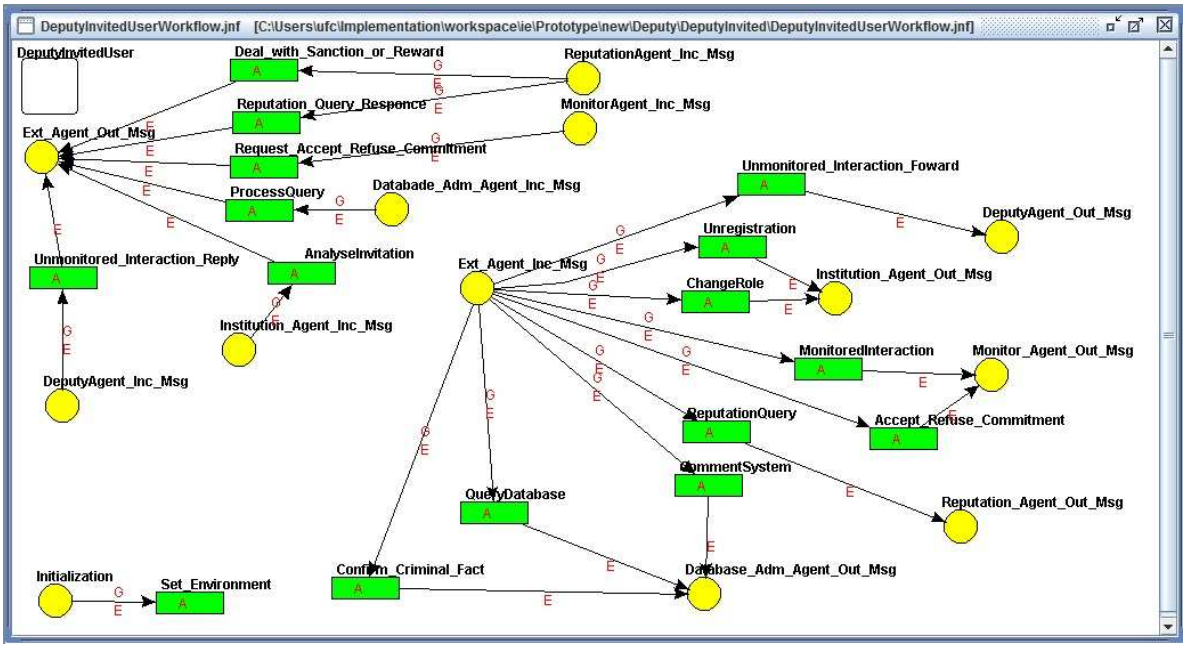


Figure 5.10. Deputy Invited User Workflow in JFern

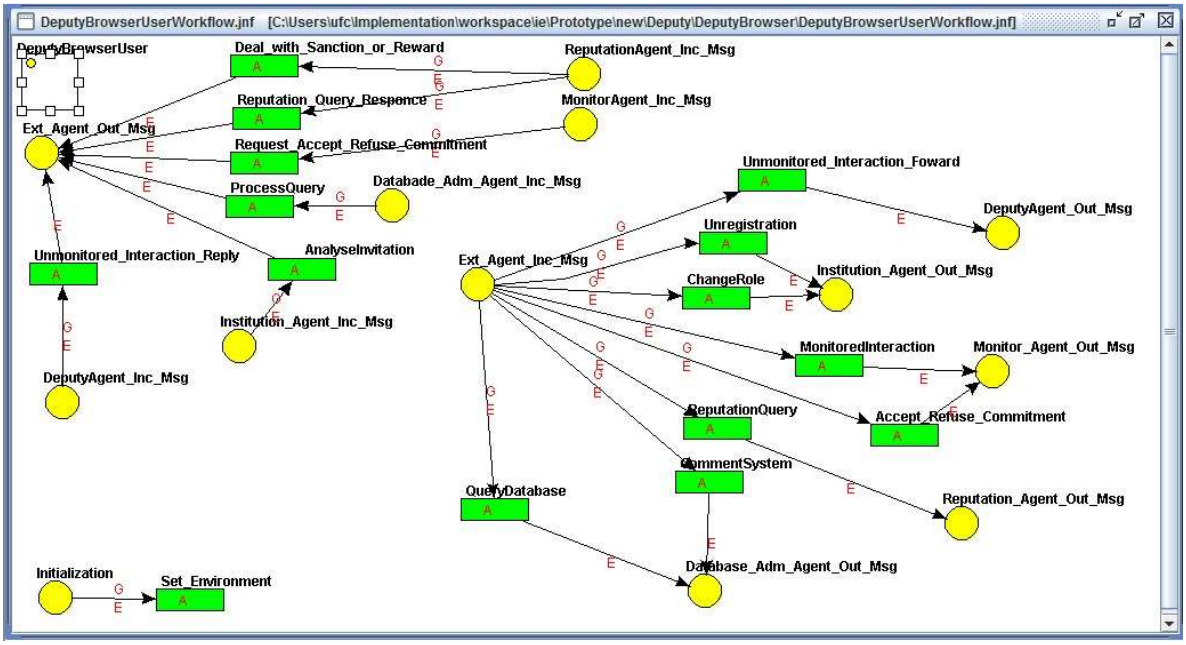


Figure 5.11. Deputy Browser User Workflow in JFern

5.4.4. Commitments

Social commitments are defined in the WikiCrimes Institutional Environment. These have external agents as debtors and the Institution Agent as creditor. The condition and content of the commitments are stored as temporal propositions defined as formulas in the Model Checker present in the Institution Agent. The commitments are managed by the Monitor Agent, which sends updates on the state of them to the Reputation Agent so that it can update the reputation score of the debtors.

This first commitment is the one that a Deputy Registered User has with the Institution Agent that it will never have a report denounced as abuse or confirmed negatively. This is a commitment template that is used for each registered user. This is a compulsory commitment that is set at registration to every external agent that registers with the institutional environment.

Creditor *InstitutionAgent*
Debtor *DeputyRegisteredUser_001*
Condition *DeputyRegisteredUser_submited_report* **and** *Report_has_status_accepted*
Content not (*Negative_confirmation_on_report_is_made* **or**
Abuse_denouce_on_report_is_made)

Related to this commitment there is a set of three count-as rules that refers to institutional facts that may result from them (listed below). Those rules express the necessary infrastructure for the institutional environment to carry on with the evaluation of the commitments. As said before, the counts-as rules are defined here to separate low-level concerns related to the functioning of the institutional environment and the high-level concerns that relate to the application domains of the institutional environment.

The counts-as rules are verified by the Institution Agent, which inserts new states in the model checker where the counts-as part of the rule is asserted as true or false. The history of the model checker will determine if the propositions that represent the condition and content of the commitments are true or false in the moment of its evaluation.

InstitutionAgent_is_Online
and *MonitorAgent_is_Online*
and *ReputationAgent_is_Online*
and *DatabaseAdministratorAgent_is_Online*
and *DeputyRegisteredUser_is_Online*
and *DatabaseAdministratorAgent_receives_RegisterCriminalFactMessage*
counts as *Report_has_status_accepted*

InstitutionAgent_receives_Negative_Confirmation_Information_from_DatabaseAdmAgent
counts as *Negative_confirmation_on_report_is_made*

and

InstitutionAgent_receives_AbuseDenounce_from_RegisteredUser
or *InstitutionAgent_receives_AbuseDenounce_from_InvitedUser*
or *InstitutionAgent_receives_AbuseDenounce_ from_Browser User*
counts as *Abuse_denouce_on_report_is_made*

For the evaluation of the counts-as rules the reports are identified by an id that is added to the institutional fact when the model present in the model checker is updated (see more details in the next chapter).

The next commitment is the one that the Deputy Invited User has with the Institution Agent that it will confirm a crime positively or negatively within 15 days from the invitation to confirm a crime.

Creditor *InstitutionAgent*
Debtor *DeputyInvitedUser*
Condition *Invitation_has_status_accepted*
Content *Confirmation_on_report_is_made_within_15_days*

In that commitment:

InstitutionAgent_is_Online
and *MonitorAgent_is_Online*
and *ReputationAgent_is_Online*
and *DatabaseAdministratorAgent_is_Online*
and *DeputyInvitedUser_is_Online*
and *InstitutionAgent_received_InvitationAcceptanceMessage*
counts as *Invitation_has_status_accepted*

In the last commitment if Invitation_has_status_accepted the Deputy Invited User can change his role with the Institution Agent to Deputy Registered User and the Monitor Agent reassigns the commitment to the Deputy Registered User.

The Deputy Registered Agents can check the reputation of agents that have registered crimes in the area where it has registered crimes and ask them to form a group of agents that register crime together in a specific area. If the agent does not check the reputation, before sending the invitation to form a group, they may get partners with bad behaviour in the group. But it is an option for the agent that is inviting others to participate in the group. It may have it own history about the behaviour of a specific external agent that has not yet developed a good reputation in the institutional environment and decide to send the invitation anyway.

Another policy that could be employed could be for the Institution Agent to check from time to time the reputation database to identify agents with a bad reputation score. These agents could be asked to perform better or leave the institutional environment.

Chapter 6

WikiCrimes Institutional Environment Walkthroughs

This chapter elaborates on examples extracted from the WikiCrimes Institutional Environment prototype. Here we describe scenarios extracted from the implementation of the specification defined in Chapter 5. They are used to demonstrate the outcomes verified when running the implemented institutional environment. This is done to highlight situations identified in the WikiCrimes System pertinent to open MAS and to show how they were handled by the Institutional Environment model defined in this thesis. The scenarios are descriptions of situations that belong to the application domain of WikiCrimes. Comments and conclusions driven from these experiments are found in the description of each scenario.

6.1. External Agent Registration

6.1.1. Scenario Description

In this scenario any given external agent contacts the Institutional Environment through the Institutional Agent and requests to be registered in the environment. This type of request triggers a sequence of events that will allow for the external agent to assume a role in the system and start interacting with other agents present in it.

The access point to the Institutional Environment is the Institution Agent. It allocates a Deputy Agent that implements the dialogs that can take place in the environment that has

the newly registered agent as initiator or participant. The CPNs depicted in Chapter 5 represent the roles that can be played by the external agents that register with the WikiCrimes Institutional Environment.

One way that the original WikiCrimes system tries to make new users join the collaborative system is by sending invitations to non-registered users to confirm criminal facts that were registered by registered users. When registering a new criminal fact the new user is asked to provide email addresses of other users, registered or not, so that the system can send invitations for those users to confirm crimes. If the users are not registered yet, they are asked to join the collaborative system (see SendInvitation transition in Figure 5.2).

We implemented this mechanism in the WikiCrimes Institutional Environment. When the agent tries to register, the email address it provides as its unique identifier in the institutional environment is checked by the Institution Agent. If it was invited to confirm crimes before, it will be asked to become a debtor of social commitments regarding the confirmation of criminal facts within 15 days of invitation. This scenario is better explored in Sections 6.2 and 6.3.

6.1.2. Activity Diagram for the Registration Process

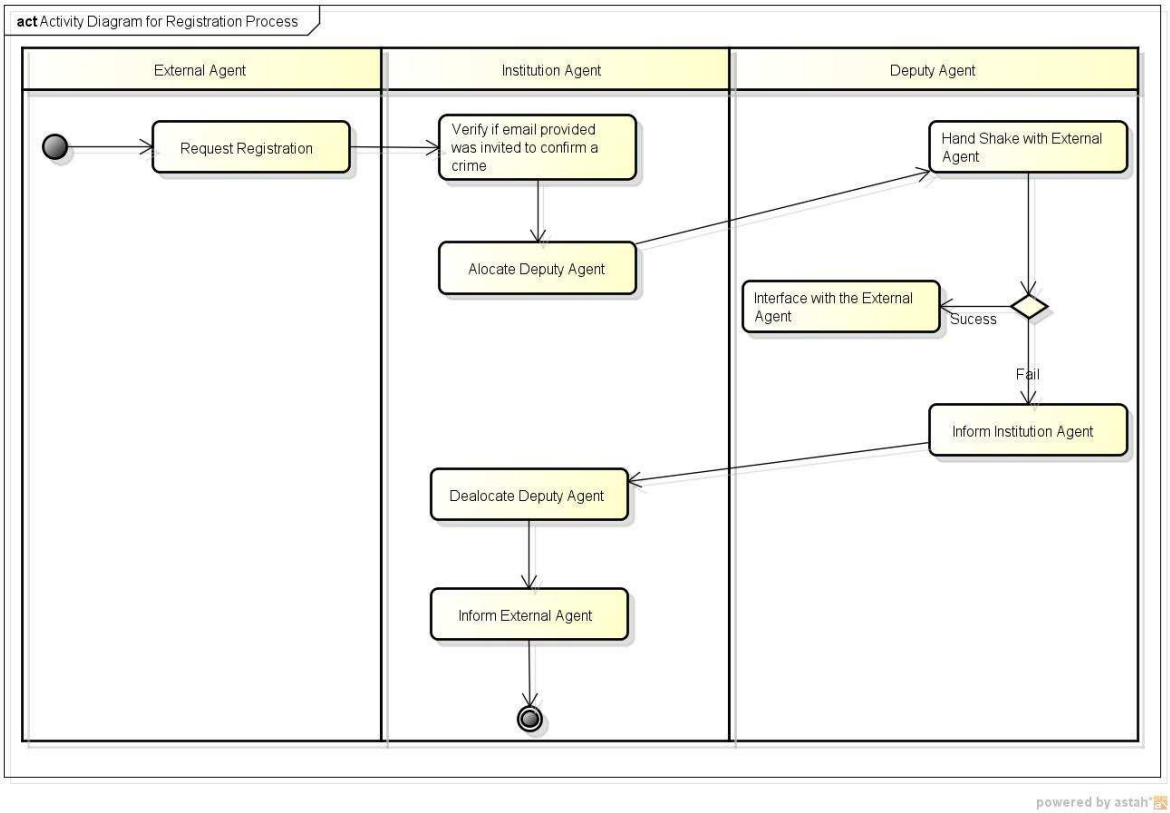


Figure 6.1. Activity Diagram for the External Agent Registration Process

6.1.3. Messages Involved in the Registration

Observe that the messages are devoted to the development of the registration. We do not worry at the moment about a more elaborated dialog between External Agents and the Institution Agent prior to the registration process. It is important to notice as well that in the WikiCrimes Institutional Environment the External Agent email address is the unique identification of this kind of agent in the environment.

Message from the External Agent requesting registration in the Institutional Environment:

```
(REQUEST
      :content    (register    :role    RegisteredUser    :external_email
'test.wikicrimes@gmail.com')
      :sender ExternalAgent@OtagoPlatform
      :receiver InstitutionAgent@OtagoPlatform
      :performative REQUEST
)
```

Message from the Institutional Agent allocating a Deputy Agent to the External Agent:

```
(INFORM
      :content        (register        :status        ok        :deputy
'DeputyRegisteredUser01@OtagoPlatform')
      :sender InstitutionAgent@OtagoPlatform
      :receiver ExternalAgent@OtagoPlatform
      :performative INFORM
)
```

Message from the Institution Agent requesting the creation of a record for the External Agent in the Reputation Data Base:

```
(REQUEST
      :content
      (create_file
            :external_email 'test.wikicrimes@gmail.com'
            :deputy 'DeputyRegisteredUser01@OtagoPlatform'
      )
      :sender InstitutionAgent@OtagoPlatform
      :receiver ReputationAgent@OtagoPlatform
      :performative REQUEST
)
```

Message from the Institution Agent requesting the creation of a social commitment for the Registered User that it commits to tell the truth in the Institutional Environment:

```
(REQUEST
      :content
      (make_commitment
            :creditor 'InstitutionAgent@OtagoPlatform'
            :debtor 'DeputyRegisteredUser01@OtagoPlatform'
            :condition 'and
      (DeputyRegisteredUser01@OtagoPlatform_submited_report
      DeputyRegisteredUser01@OtagoPlatform_Report_has_status_accepted)'
            :content 'not(or
      (DeputyRegisteredUser01@OtagoPlatform_Negative_confirmation_on_repo
      rt_is_made
      DeputyRegisteredUser01@OtagoPlatform_Abuse_denounce_on_report_is_ma
      de))'
      )
      :sender InstitutionAgent@OtagoPlatform
```

```
      :receiver MonitorAgent@OtagoPlatform
      :performative REQUEST
)
```

6.1.4. High Level CPN

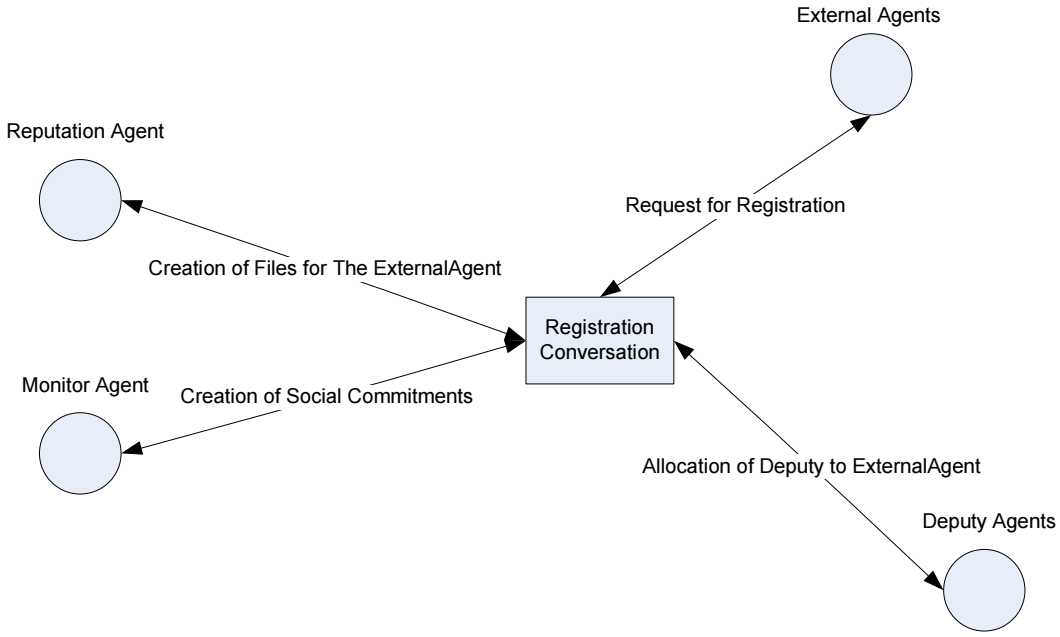


Figure 6.2. High level CPN demonstrating the conversation of agents involved in the process of registration

This CPN demonstrates the workflow of the Institutional Environment at a level where the tokens are the agents that take part in the conversation. The transition represents the actual use of the workflows that define the roles of the agents involved in the conversation[7].

6.1.5. State Diagram of the Commitment Object

The figure below demonstrates the status of the commitment object created by the process of registration of the External Agent in the WikiCrimes Institutional Environment.

[7] The high level CPNs are in this thesis for ilustration. They are going to be part of the system after future work.

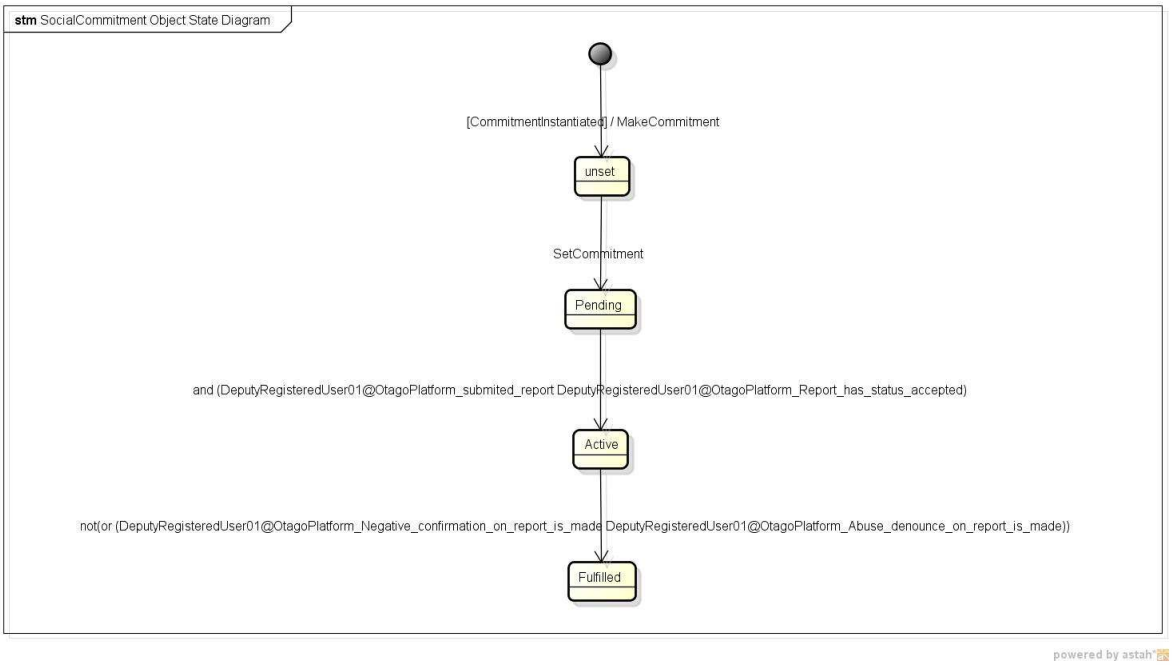


Figura 6.3. State diagram for the social commitment acquired by the External Agent when it registers with the WikiCrimes Institutional Environment

6.1.6. Model Checker Details

Here we have defined the model present in the Model Checker at the time of the registration of the External Agent to the WikiCrimes Institutional Environment.

```
m = Model(6,\
{'DeputyRegisteredUser01@OtagoPlatform_submited_report':set([]),\
'DeputyRegisteredUser01@OtagoPlatform_Report_has_status_accepted':set([])
,\
'DeputyRegisteredUser01@OtagoPlatform_Negative_confirmation_on_report_is_
made ':set([]),\
'DeputyRegisteredUser01@OtagoPlatform_Abuse_denounce_on_report_is_made':s
et([]),\
})

Condition_Commitment01 = Formula(('and',
'DeputyRegisteredUser01@OtagoPlatform_submited_report',
'DeputyRegisteredUser01@OtagoPlatform_Report_has_status_accepted'))

Content_Commitment01 = Formula(('not', ('or',
'DeputyRegisteredUser01@OtagoPlatform_Negative_confirmation_on_report_is_
made',
```

```
'DeputyRegisteredUser01@OtagoPlatform_Abuse_denounce_on_report_is_made'
)))
```

The model in this case has a number that says how many states are in the model and in curly brackets a set of propositions with a corresponding set of states where the proposition is true for very proposition.

After the evaluation of the condition and content formulas for the Commitment01 the obvious result would be the lack of states where the formulas are found true, leaving the following label for both formulas:

```
{5: [(5, False, False)], 0: [(0, False, False)], 3: [(3, False, False)],
2: [(2, False, False)], 1: [(1, False, False)], 4: [(4, False, False)]}
```

The format of this output involves subtleties of the model checker that are not relevant to our work. In brief, this shows that the formulas are false in all states.

In this scenario the propositions present in the formulas that represent the condition and content of Commitment01 are evaluated in accordance with verification of low level concerns regarding the functioning of the WikiCrimes Institutional Environment that are represented in count-as rules defined in Chapter 5. In the case of this scenario they were set to false in all the states of the model present in the model checker. States where the propositions would be true are inserted in the model at run time, which would change the outcome of the evaluation of the condition and content formulas present in the model.

6.2. Register Criminal Fact

6.2.1. Scenario Description

In this scenario any given external agent that plays the Registered User or Certifier Entity roles in the Institutional Environment sends a message to the Database Administrator Agent registering a criminal fact.

The external agent has provided on registration an email address to be used by the Institutional Environment to contact it for invitations to confirm a crime positively or negatively, or any other type of necessary communication if the external agent is not online

in the environment. When a Registered User is not online and has observation areas specified in the environment it receives emails with notification messages every time that a crime occurs in those areas or, if it is online it receives a FIPA message. The CPNs depicted in the Figures 3.7 and 5.6 specify the workflow of the Registered User Deputy.

When registering a criminal fact in the WikiCrimes Institutional Environment the agent should provide at least one email of another agent that will be invited to confirm that the criminal fact registered has occurred.

After registration the Institution Agent is in charge of sending the invitations to the agents that own the provided email addresses. If the agent is online it sends a FIPA message; if the agent is registered and is not online it sends an invitation by email and if the email address provided does not belong to any registered agent it sends an email inviting the agent to join the WikiCrimes Institutional Environment and then confirms the registered criminal fact. In any of these cases the Institution Agent sends a message to the Monitor Agent for the creation of a pre-commitment for the invited agent so that it commits itself to confirm the crime within 15 days of invitation. A message is sent by the Monitor Agent to the invited agent asking if it accepts to be the debtor of this commitment.

6.2.2. Activity Diagram for the Criminal Fact Registration Process

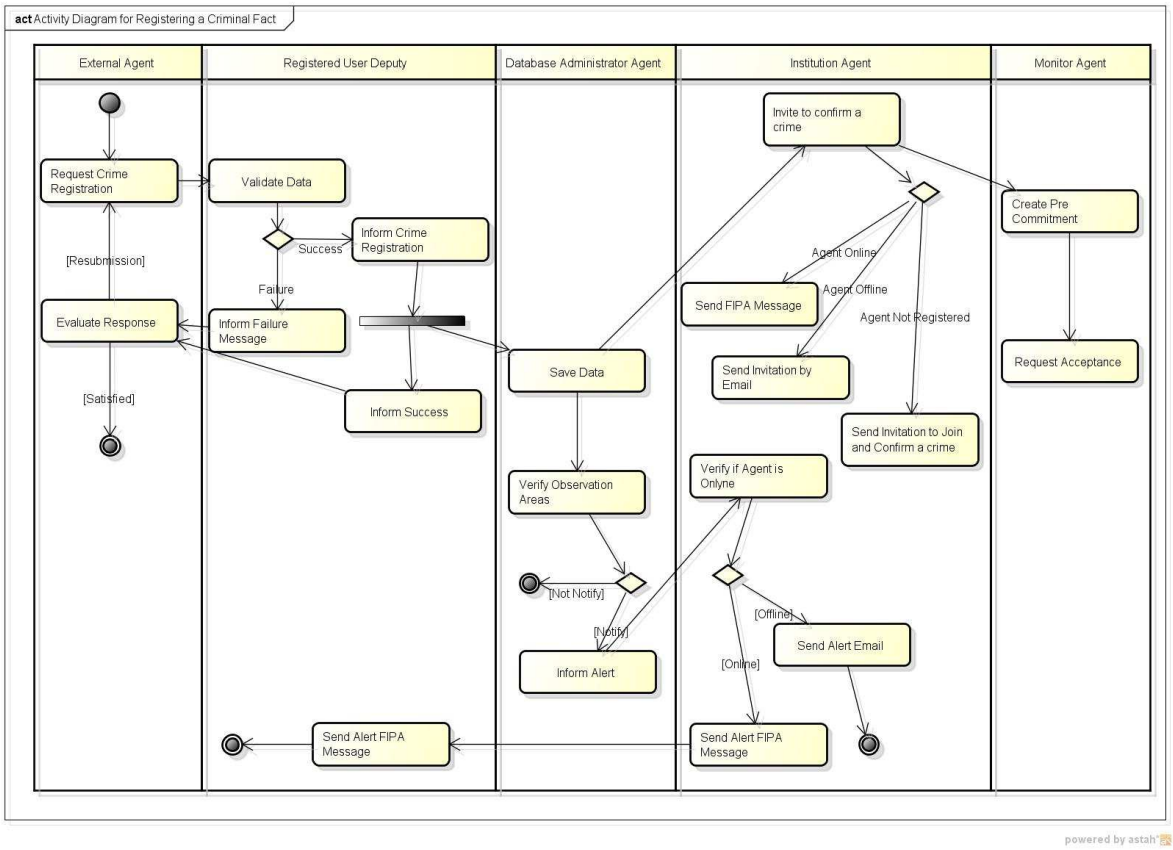


Figure 6.4. Activity Diagram for the Registration of a Criminal Fact

6.2.3. Messages Involved in the Criminal Fact Registration

There is an exchange of messages among External Agent, Registered User Deputy, Database Administrator Agent and Institution Agents so that the criminal fact registration can take place. In the sequence of messages below we assume that the External Agent provided a valid criminal fact, which is validated by the Registered User Deputy. The attributes of the criminal fact are simplified as well.

Message from the External Agent requesting criminal fact registration to the Registered User Deputy:

```
(REQUEST
      :content (register_crime :crime_type theft :latitude '999,999'
:longitude '888,888' confirming_email 'test.wikicrimes@gmail.com')
      :sender ExternalAgent@OtagoPlatform
      :receiver RegisteredUserDeputy01@OtagoPlatform
      :performative REQUEST
```

```
)
```

Message from the Registered User Deputy to the External Agent informing of the success of the criminal fact registration:

```
(INFORM
      :content (crime_registration_success)
      :sender RegisteredUserDeputy01@OtagoPlatform
      :receiver ExternalAgent@OtagoPlatform
      :performative INFORM
)
```

Message from the Registered User Deputy to the Database Administrator agent informing the criminal fact to be stored in the system database:

```
(INFORM
      :content (register_crime :crime_type theft :latitude '999,999'
:longitude '888,888' :confirming_agent 'test.wikicrimes@gmail.com')
      :sender RegisteredUserDeputy01@OtagoPlatform
      :receiver DatabaseAdministrator@OtagoPlatform
      :performative INFORM
)
```

Message from the Database Administrator Agent to the Institution Agent reporting that a crime has been registered in an observation area present in the database:

```
(INFORM
      :content
      (observation_area_alert
            (observation_area
                  :observation_area_id 3
                  :crime_id 1
                  :agent_owner_email 'registered01@email.ac.nz')
      )
      :sender DatabaseAdministrator@OtagoPlatform
      :receiver InstitutionAgent@OtagoPlatform
      :performative INFORM
)
```

Message from the Institution Agent to the Deputy Registered User informing of the observation area alert:

```
(INFORM
      :content
      (observation_area_alert
            (observation_area
                  :observation_area_id 3
                  :crime_id 1
            )
      )
      :sender InstitutionAgent@OtagoPlatform
      :receiver RegisteredUserDeputy01@OtagoPlatform
      :performative INFORM
)
```

Message from the Registered User Deputy to the External Agent informing of the observation area alert:

```
(INFORM
      :content
      (observation_area_alert
            (observation_area
                  :observation_area_id 3
                  :crime_id 1
            )
      )
      :sender RegisteredUser01@OtagoPlatform
      :receiver ExternalAgent@OtagoPlatform
      :performative INFORM
)
```

Message from the Institution Agent to the Deputy Registered User inviting it to confirm a crime:

```
(REQUEST
      :content (crime_confirmation_invitation :invitation_id 1)
      :sender InstitutionAgent@OtagoPlatform
      :receiver RegisteredUserDeputy02@OtagoPlatform
      :performative REQUEST
)
```

Message from the Institution Agent requesting creation of a pre-commitment for the Registered User so that it can evaluate if it accepts to be the debtor of a social commitment

where the content is to confirm a crime positively or negatively within 15 days of invitation:

```
(REQUEST
      :content
      (make_commitment
            :creditor 'InstitutionAgent@OtagoPlatform'
            :debtor 'DeputyRegisteredUser02@OtagoPlatform'
            :condition
      'DeputyRegisteredUser02@OtagoPlatform_Invitation_has_status_accepte
      d' :content
      'DeputyRegisteredUser02@OtagoPlatform_Confirmation_on_report_is_mad
      e_within_15_days')
      :sender InstitutionAgent@OtagoPlatform
      :receiver MonitorAgent@OtagoPlatform
      :performative REQUEST
)
```

6.2.4. High Level CPN

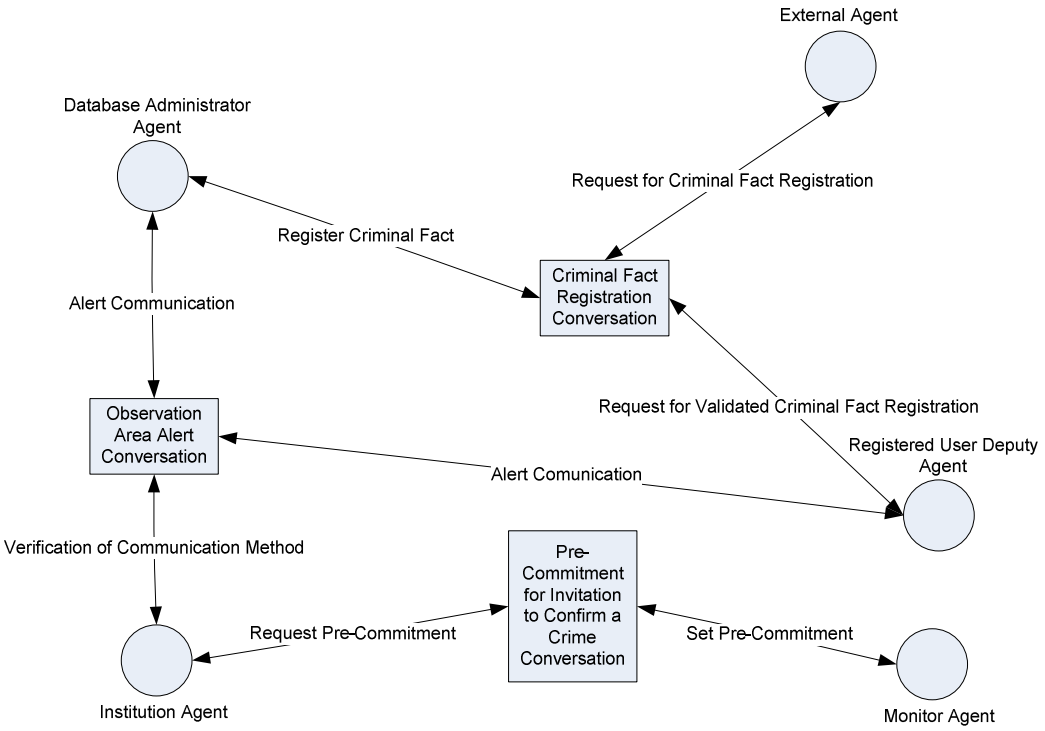


Figure 6.5. High level CPN demonstrating the conversations of agents involved in the process of a criminal fact registration

This CPN demonstrates the workflow of the Institutional Environment in a level where the tokens are the agents that take part in the conversation. The transition represents the actual use of the workflows that define the roles of the agents involved in the conversation.

6.2.5. State Diagram of the Commitment Object

There is no change of state in the commitment objects related to the agent that is inviting another to confirm a crime. There is however the creation of a pre-commitment for the agent that was invited to confirm a crime.

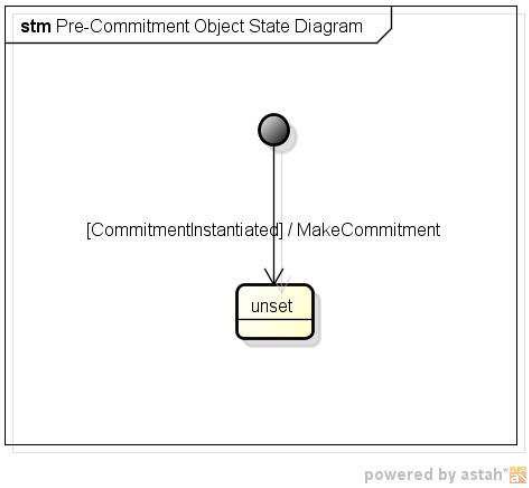


Figura 6.6. State diagram for pre-commitment created by the Monitor Agent for the agent that was invited to confirm a criminal fact

6.2.6. Model Checker Details

There is no occurrence of new facts relevant to the insertion of new states in the model checker present in the Institution Agent. Once the pre-commitment becomes a commitment the relevant facts will be inserted in the model checker.

6.3. Pre-Commitment to Confirm a Crime Acceptance

6.3.1. Scenario Description

After the Institution Agent invites an agent to confirm a crime, a pre-commitment is created by the Monitor Agent. The Monitor Agent then proposes to the invited agent to become the debtor of that pre-commitment, which will make it become a social commitment of

confirming a crime positively or negatively in the Institutional Environment within 15 days of the invitation.

If the external agents are not registered users yet, the Institution Agent, after registering the agent in the WikiCrimes Institutional Environment, will ask the Monitor Agent to create pre-commitments associated with the invitation that was sent before and the Monitor Agent will propose to the newly registered user to assume the commitment to confirm the crime within 15 days of invitation as debtor. In his case in particular, the time deadline for the commitment to become *violated* starts to count from the registration time. This is due to the nature of the invitation, which is to grow the community of agents in the WikiCrimes Institutional Environment. If the time starts to count from the invitation, as is the case with Registered User Agents, the newcomers could be penalized even before they join the Institutional Environment, which would possibly not encourage them to join.

That scenario happens when an external agent is invited to confirm a crime by email and it is not registered in the WikiCrimes Institutional Environment yet. This is a way to grow the community of agents registered in the WikiCrimes Institutional Environment, strategy similarly applied in the original WikiCrimes collaborative system.

6.3.2. Activity Diagram for the Pre-Commitment Acceptance Process

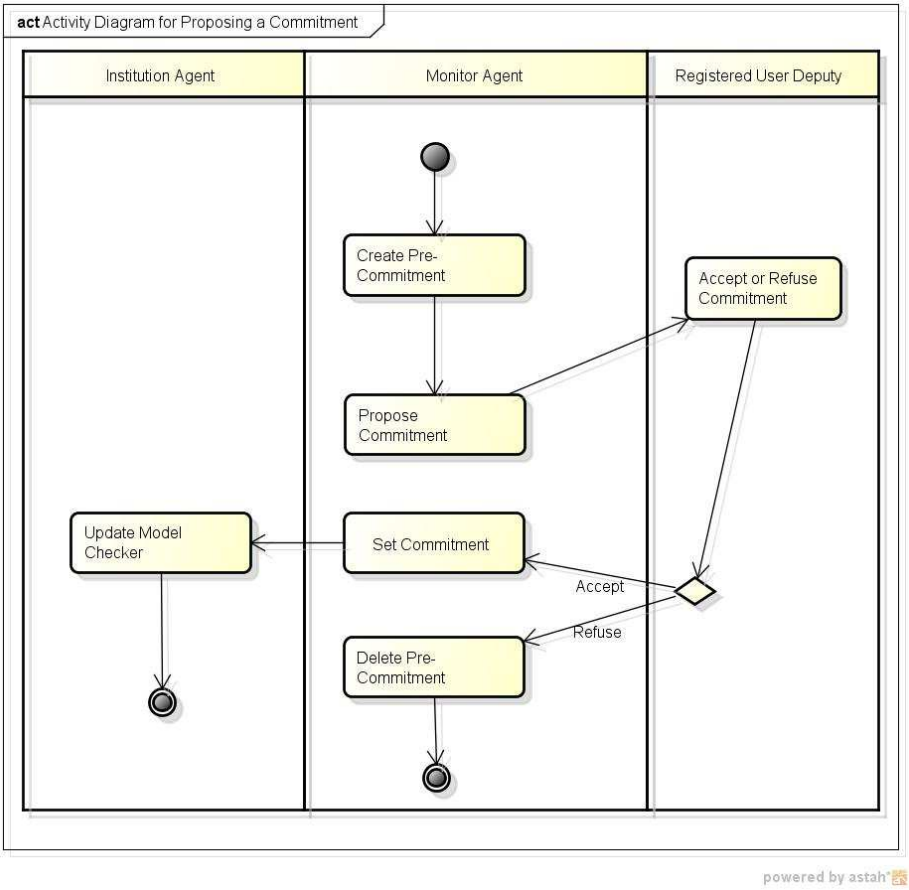


Figure 6.7. Activity Diagram for the Commitment Proposal

6.3.3. Messages Involved in the Commitment Proposal

There is an exchange of messages among the External Agent, Registered User Deputy, Institution Agent and Monitor Agent so that the commitment proposal can be accepted or refused by the external agent. In the sequence of messages below we assume that the External Agent is already registered with the WikiCrimes Institutional Environment.

Message from the Monitor Agent proposing the commitment to the Registered User Deputy to accept or refuse the commitment to confirm a crime within 15 days of invitation:

```
(REQUEST
      :content (propose_commitment (pre_commitment
            :commitment_id commitment02
            :creditor 'InstitutionAgent@OtagoPlatform'
```

```
            :condition
      'RegisteredUser03@OtagoPlatform_Invitation_has_status_accepted'
            :content
      'RegisteredUser03@OtagoPlatform_Confirmation_on_report_is_made_with
      in_15_days'))
      :sender MonitorAgent@OtagoPlatform
      :receiver RegisteredUserDeputy03@OtagoPlatform
      :performative REQUEST
)
```

Message from the Registered User Deputy to the External Agent informing of Monitor Agent proposal:

```
(REQUEST
      :content (propose_commitment (pre_commitment
            :commitment_id commitment02
            :creditor 'InstitutionAgent@OtagoPlatform'
            :condition
      'ReisteredUser03@OtagoPlatform_Invitation_has_status_accepted'
            :content
      'RegisteredUser03@OtagoPlatform_Confirmation_on_report_is_made_with
      in_15_days'))
      :sender RegisteredUserDeputy03@OtagoPlatform
      :receiver ExternalAgent@OtagoPlatform
      :performative REQUEST
)
```

Message from the External Agent to the Registered User Deputy informing of its response to the proposal:

```
(INFORM
      :content (accept_proposal :commitment_id commitment02)
      :sender ExternalAgent@OtagoPlatform
      :receiver RegisteredUserDeputy03@OtagoPlatform
      :performative INFORM
)
```

Message from the Registered User Deputy to the Monitor Agent informing of its response to the proposal:

```
(INFORM
      :content (accept_proposal :commitment_id commitment02)
```

```
      :sender RegisteredUserDeputy03@OtagoPlatform
      :receiver MonitorAgent@OtagoPlatform
      :performative INFORM
)
```

Message from the Monitor Agent to the Institution Agent updating the Model Checker:

```
(REQUEST
      :content
      (update_model_insert
            :commitment Commitment01
            :condition
      'DeputyRegisteredUser03@OtagoPlatform_Invitation_has_status_accepte
      d'
            :content
      'DeputyRegisteredUser03@OtagoPlatform_Confirmation_on_report_is_mad
      e_within_15_days')
      :sender MonitorAgent@OtagoPlatform
      :receiver InstitutionAgent@OtagoPlatform
      :performative REQUEST
)
```

6.3.4. High Level CPN

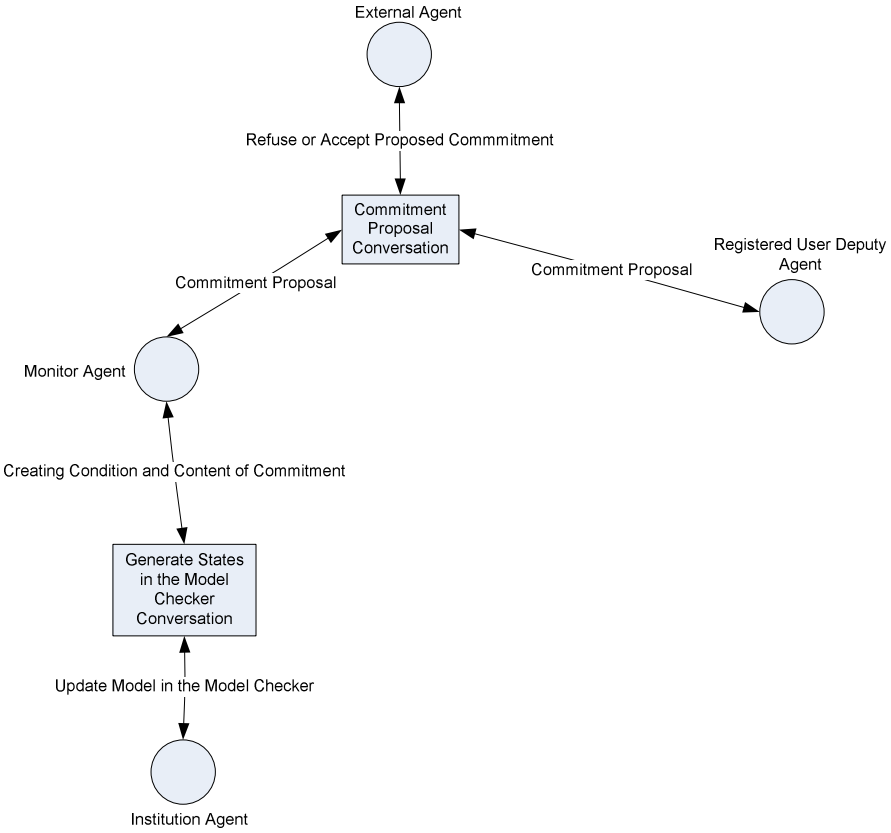


Figure 6.8. High level CPN demonstrating the conversations of agents involved in the process of commitment proposal

This CPN demonstrates the workflow of the Institutional Environment at a level where the tokens are the agents that take part in the conversation. The transition represents the actual use of the workflows that define the roles of the agents involved in the conversation.

6.3.5. State Diagram of the Commitment Object

The state of the commitment object is set to *pending* and the pre-commitment becomes a commitment.

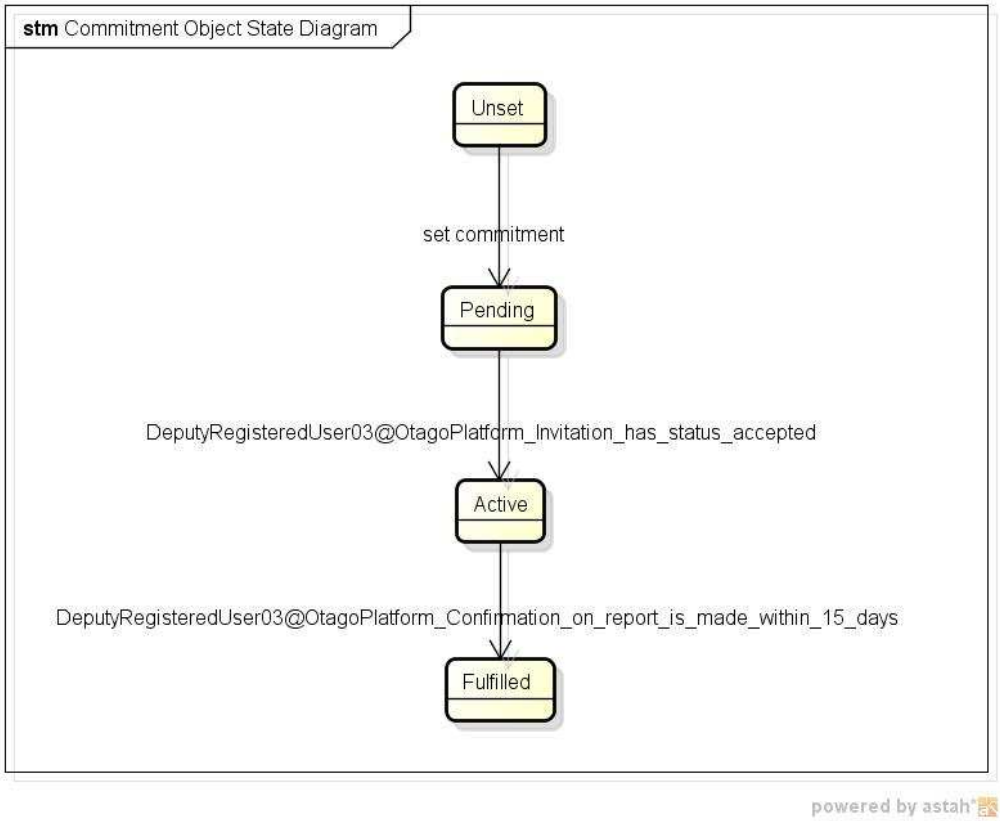


Figura 6.9. State diagram updating a pre-commitment to a commitment for the agent that was invited to confirm a criminal fact and has accepted the proposal

6.3.6. Model Checker Details

The model in the Model Checker is updated with the insertion of the content and condition of the newly created commitment object.

```
m = Model(6,\
{'DeputyRegisteredUser01@OtagoPlatform_submited_report':set([]),\
'DeputyRegisteredUser01@OtagoPlatform_Report_has_status_accepted':set([])
,\
'DeputyRegisteredUser01@OtagoPlatform_Negative_confirmation_on_report_is_
made':set([]),\
'DeputyRegisteredUser01@OtagoPlatform_Abuse_denounce_on_report_is_made_ev
er':set([]),\
'DeputyRegisteredUser03@OtagoPlatform_Invitation_has_status_accepted':set
([4,5]),\
'DeputyRegisteredUser03@OtagoPlatform_Confirmation_on_report_is_made_with
in_15_days':set([5]),\
})
```

```
Condition_Commitment03 =
Formula(('DeputyRegisteredUser03@OtagoPlatform_Invitation_has_status_acce
pted'))

Content_Commitment03 =
Formula('DeputyRegisteredUser03@OtagoPlatform_Confirmation_on_report_is_m
ade_within_15_days'))
```

After the evaluation of the condition and content formulas for the Commitment03 when the condition is true in states 4 and 5 and the content is true is state 5 we have the following label for the formulas:

Condition:

```
{5: [(5, True, True)], 0: [(0, False, False)], 3: [(3, False, False)], 2:
[(2, False, False)], 1: [(1, False, False)], 4: [(4, True, True)]}
```

Content:

```
{5: [(5, True, True)], 0: [(0, False, False)], 3: [(3, False, False)], 2:
[(2, False, False)], 1: [(1, False, False)], 4: [(4, False, False)]}
```

In this scenario the propositions present in the formulas that represent the condition and content of Commitment03 are evaluated according to verification of low level concerns regarding the functioning of the WikiCrimes Institutional Environment that are represented in count-as rules defined in Chapter 5. In the case of this scenario they were set to true to make the proposition Invitatation_has_status_accepted become true in the states 4 and 5, and the proposition Confirmation_on_report_is_made_within_15_days was set to true in state 5 because the low level calculation of the deadline did not find that it had expired, which means that the RegisteredUserDeputy02 has made the confirmation of the criminal fact within 15 days of invitation.

6.4. Confirm a Crime Negatively

6.4.1. Scenario Description

In this scenario a Registered User confirms a crime negatively, which means that he believes that the information about the crime is not accurate or that it did not happen. This

situation will cause the state change of a social commitment that states the Registered User will tell the truth always (see below), which means that the commitment should not have a negative confirmation nor be denounced as abuse.

Creditor *InstitutionAgent*
Debtor *DeputyRegisteredUse01*
Condition *DeputyRegisteredUser_submited_report* **and** *Report_has_status_accepted*
Content not (*Negative_confirmation_on_report_is_made* **or**
Abuse_denouce_on_report_is_made)

The way it works is that when the Registered User registers it will be the debtor of this commitment compulsorily; it is a requirement for registration. Therefore the commitment should be in the state *fulfilled* always. It cannot be *violated* because it does not have a deadline associated with it but it can be *active* with content false. If this happens, it is because its reputation in telling the truth should be questioned and the Reputation Agent should be actioned to re-calculate its reputation score.

Therefore the commitment is set to *fulfilled* from the beginning and whenever it is only *active* it is because the Registered User did not tell the truth in the reporting of a crime and the Reputation Agent is contacted to calculate a new reputation score for that agent.

Immediately after the insertion of a state where that commitment is not *fulfilled* the propositions condition is made true again. That is because we assume that the agent should be telling the truth and committed a mistake, even though he is punished in reputation points for that.

6.4.2. Activity Diagram for the Negative Confirmation

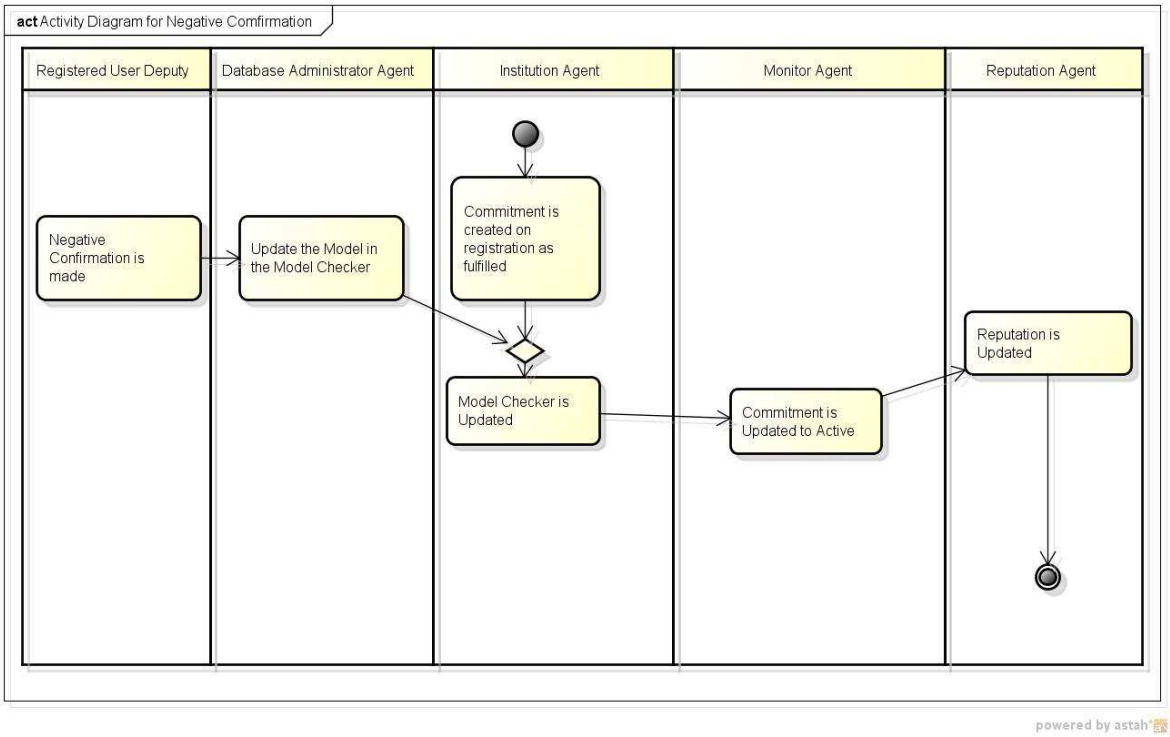


Figure 6.10. Activity Diagram for the negative confirmation of a Criminal Fact

6.4.3. Messages Involved in the Negative Confirmation

There is an exchange of messages among the External Agent, Registered User Deputy, Institution Agent and Monitor Agent and Reputation Agent so that the commitment to tell the truth where the Registered User is a debtor changes state from *fulfilled* to *active*. In the sequence of messages below we assume that the External Agent has already made a valid report confirming negatively a crime registered by another Registered User. We consider here only the messages for the update of commitment and reputation score.

Message from the Registered User Deputy confirming a crime negatively to the Database Administrator Agent:

```
(REQUEST
      :content (crime_confirmation
            :crime_id 1
            :type negative
            :crime_reporter 'RegisteredUserDeputy01@OtagoPlatform'
            )
      :sender RegisteredUserDeputy03@OtagoPlatform
```

```
      :receiver DatabaseAdministratorAgent@OtagoPlatform
      :performative REQUEST
)
```

Message from the Database Administrator Agent to the Institution Agent requesting update in the Model of the Model Checker:

```
(REQUEST
      :content (update_modelcheker
                        :truthvalue true:fact
      'DeputyRegisteredUser01@OtagoPlatform_Negative_confirmation_on_repo
      rt(2)_is_made'
            )
            )
      :sender DatabaseAdministratorAgent@OtagoPlatform
      :receiver InstitutionAgent@OtagoPlatform
      :performative REQUEST
)
```

Message from the Institution Agent to the Monitor Agent reporting change in the Model present in the Model Checker changing the state of the evaluation result of the formula that represents the content of the commitment the Registered User has to tell the truth:

```
(REQUEST
      :content (set_commitment
            :updates_(Commitment01 active Commitment02 fulfilled
      Commitment03 violated)
            )
      :sender InstitutionAgent@OtagoPlatform
      :receiver MonitorAgent@OtagoPlatform
      :performative REQUEST
)
```

Message from the Monitor Agent to the Reputation Agent informing requesting update in reputation score:

```
(REQUEST
      :content (update_reputation
            :commitment_evaluation content_false
            :debtor DeputyRegisteredUser01@OtagoPlatform
            )
      :sender MonitorAgent@OtagoPlatform
```

```
      :receiver ReputationAgent@OtagoPlatform
      :performative REQUEST
)
```

6.4.4. High Level CPN

This CPN demonstrates the workflow of the Institutional Environment at a level where the tokens are the agents that take part in the conversation. The transition represents the actual use of the workflows that define the roles of the agents involved in the conversation.

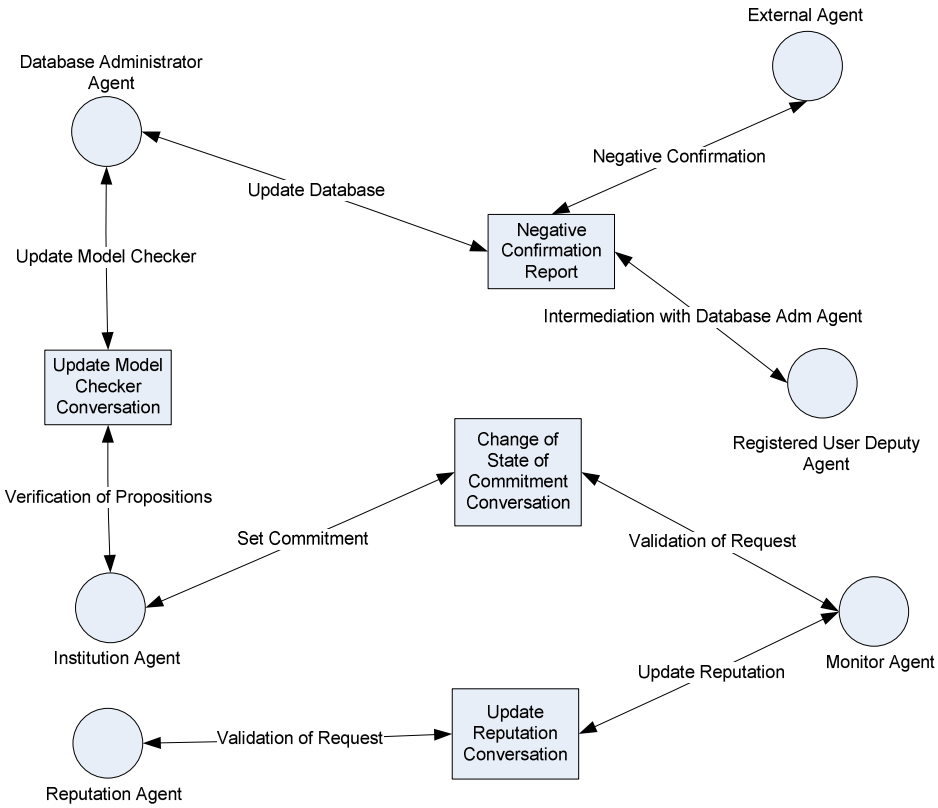


Figure 6.11. High level CPN demonstrating the conversations of agents involved in the process of negative confirmation

6.4.5. State Diagram of the Commitment Object

The state of the commitment object is set to violated, and a new social commitment is created in the Monitor Agent to keep punishing the Registered User if it keeps being confirmed negatively or denounced because of abuse.

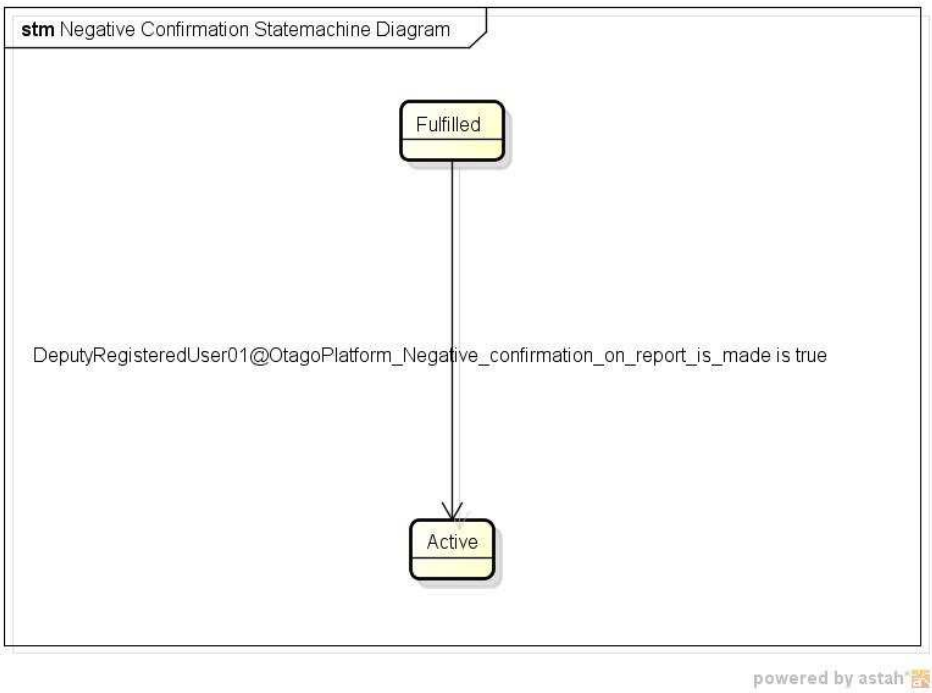


Figura 6.12. State diagram updating the commitment state due to a change in the truth value of a proposition present in its content formula

6.4.6. Model Checker Details

The model in the Model Checker is updated with the insertion of a state where the proposition `Negative_confirmation_on_report_is_made` for the Registered User Deputy is set to true. The update in the commitment object is made and then the proposition is set to true again in the following state to ensure that the Institutional Environment will consider the agent still committed to tell the truth. In the code below the proposition is not true in the state 4 but then true in the following state 5.

```
m = Model(6,\
{'DeputyRegisteredUser01@OtagoPlatform_submited_report':set([0,1,2,3,4,5]
),\
'DeputyRegisteredUser01@OtagoPlatform_Report_has_status_accepted':set([0,
1,2,3,4,5]),\
'DeputyRegisteredUser01@OtagoPlatform_Negative_confirmation_on_report_is_
made':set([3]),\
'DeputyRegisteredUser01@OtagoPlatform_Abuse_denounce_on_report_is_made_ev
er':set([]),\
'DeputyRegisteredUser03@OtagoPlatform_Invitation_has_status_accepted':set
([4,5]),\
```

```
'DeputyRegisteredUser03@OtagoPlatform_Confirmation_on_report_is_made_with
in_15_days':set([5]),\
})

Condition_Commitment01 = Formula(('and',
'DeputyRegisteredUser01@OtagoPlatform_submited_report',
'DeputyRegisteredUser01@OtagoPlatform_Report_has_status_accepted'))


Content_Commitment01 = Formula(('not', ('or',
'DeputyRegisteredUser01@OtagoPlatform_Negative_confirmation_on_report_is_
made',
'DeputyRegisteredUser01@OtagoPlatform_Abuse_denounce_on_report_is_made_ev
er' )))
```

After the evaluation of the condition formula for the Commitment01 the result is true for all the states present in the model:

```
{5: [(5, True, True)], 0: [(0, True, True)], 3: [(3, True, True)], 2:
[(2, True, True)], 1: [(1, True, True)], 4: [(4, True, True)]}
```

After the evaluation of the content formula for the Commitment01, the result is false for the state 3 of the model, because in it the proposition `Negative_confirmation_on_report_is_made` is set to true. In the states before and after the 3 the evaluation of the condition formula is true because the proposition negative reports are not being made nor denounces of abuse related to the crimes reported by agent `DeputyRegisteredUser01@OtagoPlatform`:

```
{5: [(5, True, True)], 0: [(0, True, True)], 3: [(3, False, False)], 2:
[(2, True, True)], 1: [(1, True, True)], 4: [(4, True, True)]}
```

After the evaluation and change of state of the social commitment to *active* the model is updated when a new report is made by the Registered User.

6.5. Confirm a Crime Positively

6.5.1. Scenario Description

In this scenario a Registered User confirms a crime positively, which means that he or she believes that the information about the crime is accurate and that it did happen. This situation will not cause a change to the state of the social commitment that states the Registered User will always tell the truth.

6.5.2. Activity Diagram for Positive Confirmation

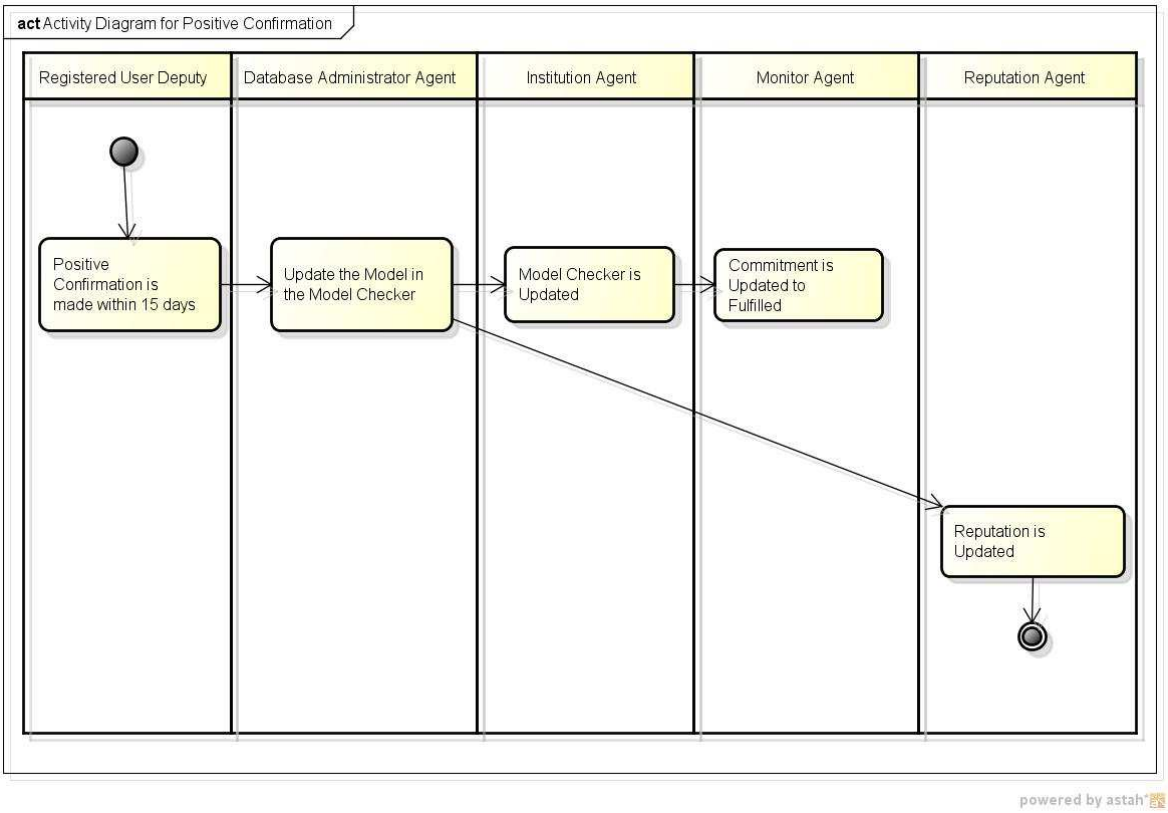


Figure 6.13. Activity Diagram for positive confirmation

6.5.3. Messages Involved in the Positive Confirmation

There is an exchange of messages among the External Agent, Registered User Deputy, Institution Agent and Monitor Agent and Reputation Agent so that the commitment to confirm a crime within 15 days of invitation where the Registered User is a debtor changes the state from *active* to *fulfilled* or violated. In the sequence of messages below we assume

that the External Agent has already made a valid report confirming positively a crime registered by another Registered User within 15 days of invitation. When the invitation is made, a data structure in the Institution Agent stores the deadlines associated with propositions before new states are inserted in the Model Checker the deadlines are checked (see transition SendInvitation from Figure 5.2). We consider here only the messages for the update of commitment and reputation score (see transition Confirm_CriminalFact on Figure 5.5), since the process of checking the deadline is coded in the Institution Agent.

Message from the Registered User Deputy confirming a crime positively to the Database Administrator Agent:

```
(REQUEST
      :content (crime_confirmation
            :crime_id 2
            :type positive
            :crime_reporter 'RegisteredUserDeputy01@OtagoPlatform'
            )
      :sender RegisteredUserDeputy03@OtagoPlatform
      :receiver DatabaseAdministratorAgent@OtagoPlatform
      :performative REQUEST
)
```

Message from the Database Administrator Agent to the Monitor Agent requesting update in the Model of the Model Checker:

```
(REQUEST
      :content (update_modelchecker
            :truthvalue true
            :fact
      'DeputyRegisteredUser03@OtagoPlatform_Confirmation_on_report(1)_is_
      made_within_15_days'
            )
            )
      :sender DatabaseAdministratorAgent@OtagoPlatform
      :receiver InstitutionAgent@OtagoPlatform
      :performative REQUEST
)
```

Observe that in this last message the report is identified with an id. That id is to check the deadline associated with this proposition in the Institution Agent.

Message from the Institution Agent to the Monitor Agent reporting change in the Model present in the Model Checker changing the state of the evaluation result of the formula that represents the content of the commitment that the Registered User has to tell the truth:

```
(REQUEST
      :content (set_commitment
            :updates_(Commitment01 active Commitment02 fulfilled
      Commitment03 fulfilled)
            )
      :sender InstitutionAgent@OtagoPlatform
      :receiver MonitorAgent@OtagoPlatform
      :performative REQUEST
)
```

Message from the Database Administrator Agent to the Reputation Agent requesting an update in reputation score:

```
(REQUEST
      :content (update_reputation
            :type positive
            :agent 'RegisteredUserDeputy03@OtagoPlatform'
            )
      :sender DatabaseAdmAgent@OtagoPlatform
      :receiver ReputationAgent@OtagoPlatform
      :performative REQUEST
)
```

6.5.4. High Level CPN

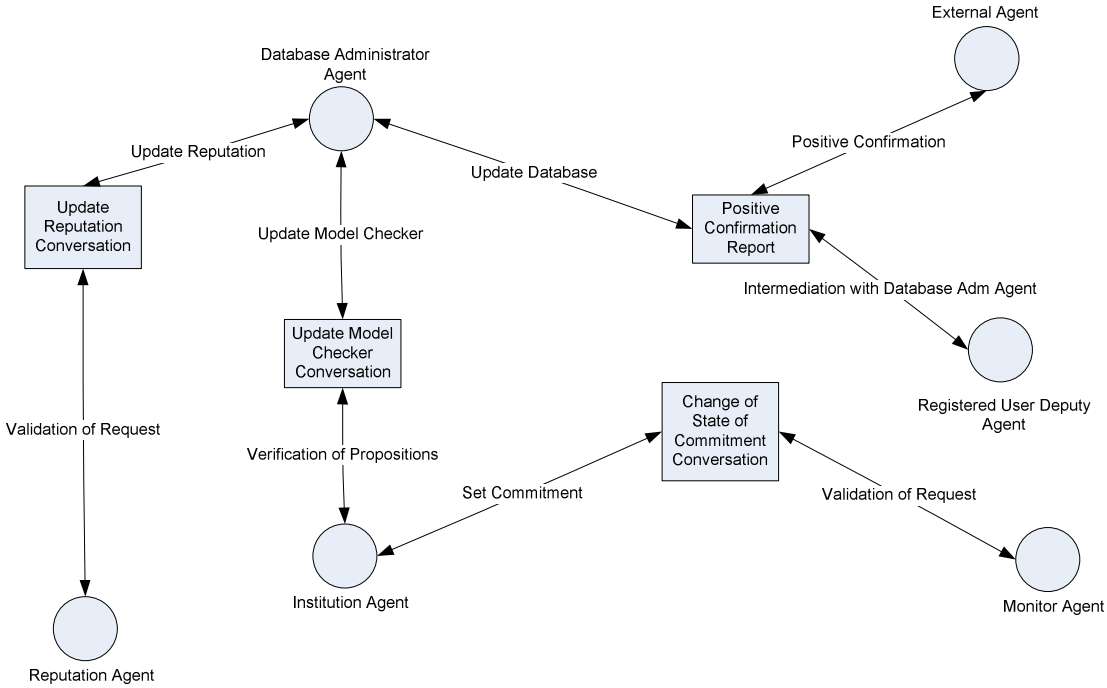


Figure 6.14. High level CPN demonstrating the conversations of agents involved in the process of positive confirmation

This CPN demonstrates the workflow of the Institutional Environment in a level where the tokens are the agents that take part in the conversation. The transition represents the actual use of the workflows that define the roles of the agents involved in the conversation.

6.5.5. State Diagram of the Commitment Object

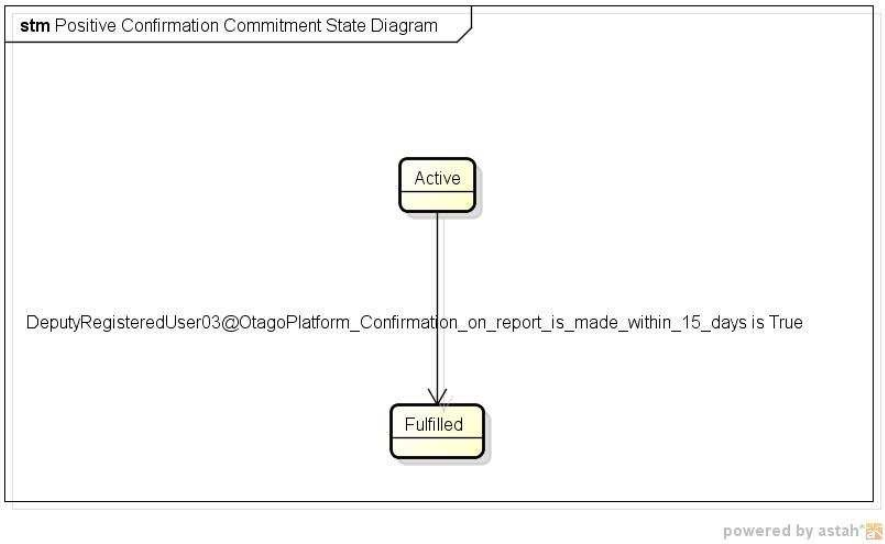


Figura 6.15. State diagram updating the commitment state due to a change in the truth value of a proposition present in its content formula

6.5.6. Model Checker Details

The model in the Model Checker is updated with the insertion of the content and condition of the newly created commitment object.

```
m = Model(6,\
{'DeputyRegisteredUser01@OtagoPlatform_submited_report':set([]),\
'DeputyRegisteredUser01@OtagoPlatform_Report_has_status_accepted':set([])
,\
'DeputyRegisteredUser01@OtagoPlatform_Negative_confirmation_on_report_is_
made':set([]),\
'DeputyRegisteredUser01@OtagoPlatform_Abuse_denounce_on_report_is_made_ev
er':set([]),\
'DeputyRegisteredUser03@OtagoPlatform_Invitation_has_status_accepted':set
([4,5]),\
'DeputyRegisteredUser03@OtagoPlatform_Confirmation_on_report_is_made_with
in_15_days':set([5]),\
})

Condition_Commitment03 =
Formula(('DeputyRegisteredUser03@OtagoPlatform_Invitation_has_status_acce
pted'))
```

```
Content_Commitment03 =
Formula('DeputyRegisteredUser03@OtagoPlatform_Confirmation_on_report_is_m
ade_within_15_days'))
```

After the evaluation of the condition and content formulas for the Commitment03 when the condition is true in states 4 and 5 and the content is true is state 5 we have the following label for the formulas:

Condition:

```
{5: [(5, True, True)], 0: [(0, False, False)], 3: [(3, False, False)], 2:
[(2, False, False)], 1: [(1, False, False)], 4: [(4, True, True)]}
```

Content:

```
{5: [(5, True, True)], 0: [(0, False, False)], 3: [(3, False, False)], 2:
[(2, False, False)], 1: [(1, False, False)], 4: [(4, False, False)]}
```

These label means that the commitment holds and that the confirmation was made on state 5. It is important to notice that the deadline was checked before the proposition that represents the content was inserted as true in the state 5 (see transition Update_Modelchecker_by_DBAdmAgent in Fugure 5.2).

6.6. Freedom of Speech

6.6.1. Scenario Description

Registered Agents can develop unmonitored interactions with each other in the WikiCrimes Institutional Environment. Basically the external agents could use the Institutional environment to meet and develop the predefined conversations using the Institutional Acts defined here or any other kind of conversation they want and are able to develop.

6.6.2. Activity Diagram for Free Speech

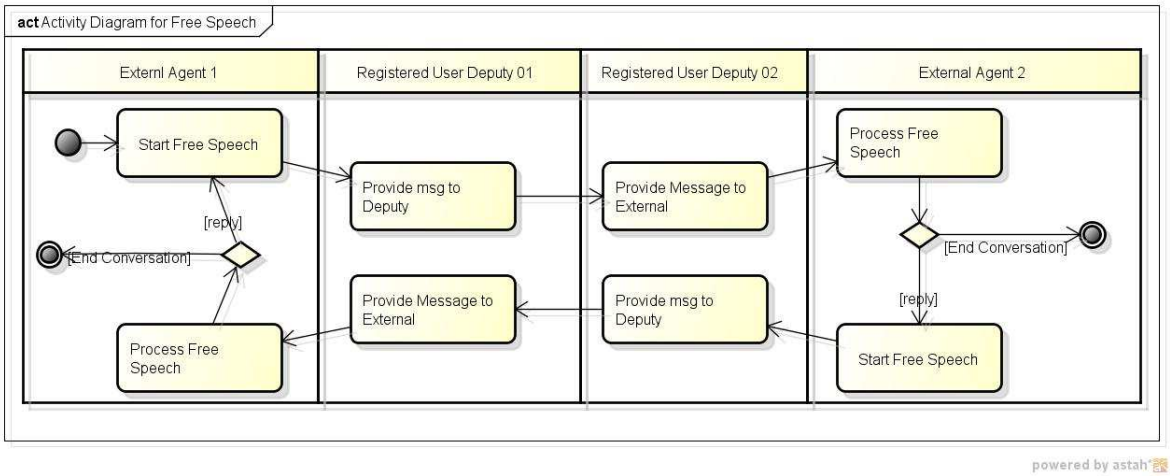


Figure 6.16. Activity Diagram for Free Speech

6.6.3. Messages Involved in the Free Speech

In this scenario there is an exchange of messages among External Agents via their respective Deputy Agents. The external agents can exchange messages out of the Institutional Environment, but even for that they would have to know about each other first and the free speech facility enables this kind of contextualized meeting point. After all, in the Institutional environment the agents are only represented by Deputy Agents and they do not know each other until they exchange contacts through the free speech facility.

Message from the External User to a Registered User Deputy asking to use the Free Speech Facility:

```
(REQUEST
      :content (free_speech
            :receiver 'RegisteredUserDeputy02@OtagoPlatform'
            :content 'hello!'
            )
      :sender ExternalAgent01@OtagoPlatform
      :receiver RegisteredUserDeputy01@OtagoPlatform
      :performative REQUEST
)
```

Message from the Registered User Deputy to the other Registered User Deputy delivering the free speech request:

```
(REQUEST
      :content (free_speech
            :sender 'RegisteredUserDeputy01@OtagoPlatform'
            :content 'hello!'
            )
      :sender RegisteredUserDeputy01@OtagoPlatform
      :receiver RegisteredUserDeputy02@OtagoPlatform
      :performative REQUEST
)
```

Message from the Registered User Deputy to the other External Agent delivering the free speech message:

```
(REQUEST
      :content (free_speech
            :sender 'RegisteredUserDeputy01@OtagoPlatform'
            :content 'hello!'
            )
      :sender RegisteredUserDeputy02@OtagoPlatform
      :receiver ExternalAgent02@OtagoPlatform
      :performative REQUEST
)
```

Message from the External User to its Deputy replying to the Free Speech message:

```
(INFORM
      :content (free_speech
            :receiver 'RegisteredUserDeputy01@OtagoPlatform'
            :content 'Ola!'
            )
      :sender ExternalAgent02@OtagoPlatform
      :receiver RegisteredUserDeputy02@OtagoPlatform
      :performative REQUEST
)
```

Note that the external agents can use the free speech to know each other addresses and stabilish conversations directly out of the institutional environment. But it is not always the case that an external agent wants to inform its real address. The free speech allows for the exchange of unpredicted conversations in the institutional context with the agents having the option to be identified only by their deputy references.

6.6.4. High Level CPN

This CPN demonstrates the workflow of the Institutional Environment at a level where the tokens are the agents that take part in the conversation. The transition represents the actual use of the workflows that define the roles of the agents involved in the conversation.

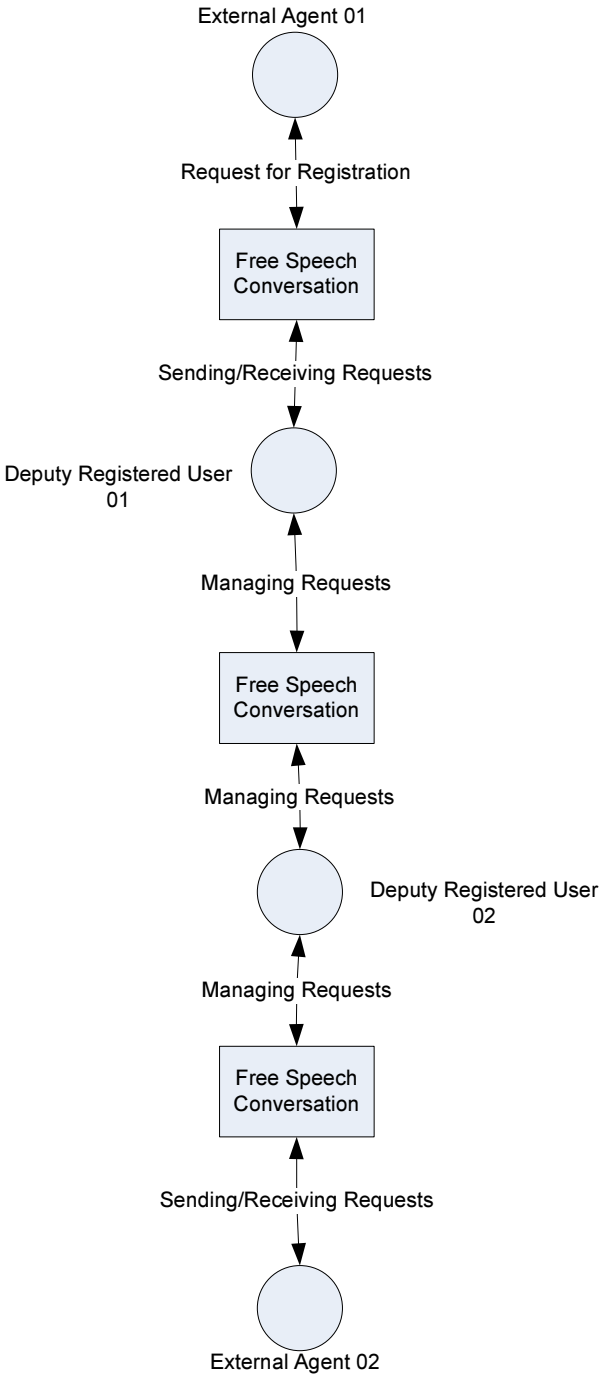


Figure 6.17. High level CPN demonstrating the conversations of agents involved in the process of free speech

Chapter 7

Conclusions

We have described in this thesis a model for the implementation of open MAS known as Institutional Environments. For the development of this model we were inspired by other approaches found in the research community and that are described in Chapter 2. Here we draw some conclusions and future directions for our research.

7.1. Software Engineering Issues

An important aspect of our approach is that we do not use finite state machines to represent an electronic institution and the conversations in the institution. Our approach is the use of CPNs to represent the institution's normative space and conversation space, which includes the roles played by agents in the institution. By that we seek the use of a formalism defined over concurrency concepts and powerful semantics relating states and actions. In our workflows we can accommodate concurrent interactions, a feature that is not possible through the use of finite state machines.

Our aim has been to enable real distributed environments to be built observing institutional norms. The strict control associated with conventional interface agents of other approaches is too restrictive and more middleware-like than agent-like. That approach was taken to allow more autonomy for the agents registered in the institutional environment, since they may not behave as predicted at the design time of the Institutional Environment. Embedded in the Institutional Environment model is the capacity of developing unpredicted behaviour that may be very pertinent to more autonomous agents with a higher capacity for reasoning.

The union of the workflows of the Institutional Environment is a comprehensive view of the system as a whole and that comprises many different levels of abstraction. The workflows are not CPNs that reside in a more centralized middleware and are managed by system agents; instead they represent system agents together with the agents registered in the Institutional Environment. Every single agent is an instance of a distributed environment and they can run on their own – they could run even in different machines. The agents interoperate through the use of FIPA messages and the workflows define the interaction protocols they implement for conversations. The System Agents offer the necessary support for the composition of the system as an Institutional Environment.

The tools being used are Opal (Nowostawski, Purvis and Cranefield, 2002) and JFern (Nowostawski, 2009). Opal is an agent development framework developed on top of the FIPA specifications. Its main characteristic is the multi-level approach for the definition of complex agents composed of less complex micro-agents, which are reactive agents in an agent-oriented programming fashion.

Our initial Java approach for the CPN inscriptions has been proved to be a bit challenging for the system designer. High level concepts from the agent-oriented paradigm are mixed with lower level concepts from the object-oriented model. As subnets give us the power of abstraction it is important that the CPN designer has a more abstract way to represent net inscriptions, without having to worry about object-oriented programming when he is developing an institutional environment.

CPN ML (Jensen, 1992) is a language used to write CPN inscriptions and declarations. It was defined based on Standard ML (Milner, 1990) and adapts its syntax towards CPN definition and formalization. In an attempt to change the attention from the institutional environment engineer towards the higher level concepts involved when implementing open multiagent systems leaving concerns, such as pattern matching and search implementation in its right level of abstraction, we are similarly making use of a higher level language that was integrated with JFern. We have made experiments with Scheme (Dybvig, 2003), Lua (Ierusalimschy, De Figueiredo and Celes, 2005) and Clojure (Halloway, 2009). Clojure was chosen because it has excellent sequence and list manipulation similar to Scheme, plus it is incorporated into the Java programming architecture, which is very well suited for the integration with JFern, which is also developed in Java.

We believe that the combination of the formal semantics of CPN and the Clojure language, which is a high level language, will offer a higher level tool suited to the development of institutional environments with different levels of abstraction helping the system engineers to develop and analyze distributed open multi-agent systems as normative spaces.

An important development in the area of electronic institutions has been done by Arcos et al. (2005), Minsky and Ungureanu (2000) and Ricci and Ominici (2003). Even though we, like Arcos et al. (2005), have the goal of implementing a socially-centred framework to develop open MAS, our approach differs in many aspects from theirs. We adapt the work done in Colombetti, Fornara and Verdicchio (2002) and Singh (1999) in the definition of our commitment-based infrastructure and use it to define objects whose active state changes during interactions of associated agents.

In our architecture the three types of operational agents use the concept of trust to better accommodate the openness that we claim to have in our model, since we do not rely on interface agents to implement norms in our institutional environment. Instead of having a special operational agent, such as the governor (Arcos et al., 2005) and controller (Minsky and Ungureanu, 2000), which forces the agents to comply with the interactions in the institution, we use the concept of trust among agents and normative spaces so that different levels of security and other important constraints can be implemented in a way that the agent itself will decide which strategy to use to comply with the societal norms and avoid being penalized by losing reputation points. At first, agents might lose access to system resources, and then gradually be expelled from the institutional environment.

Past normative models for electronic institutions (Aldewereld et al., 2006) are constrained to a closed environment built using specialized middleware. Our aim is to have these models analysed in a distributed institutional environment visualized as a normative space.

7.2 Discussion

We believe that a truly open MAS should allow for the participation of any kind of self-interested agents. Who knows and can predict completely what the self interests of the agents are? It is a big simplification to assume that they will always behave badly and force them to behave only in predicted ways. This might be very safe on one hand but on the other it limits the very intrinsic characteristic of the agent that is its autonomy. Completely

constrained systems can be implemented by other paradigms, such as the object-oriented, but not the multiagent one.

Therefore, we also believe that an open MAS should implement mechanisms to deal with the performance of unauthorized actions, or unpredicted acts. In an attempt to model open MAS using the human model to organize their societies, our approach is to introduce mechanisms that allow for the classification of different kinds of agents in the open MAS and allow, as well, for the use of any conversation strategy the agents need or want to use. This feature is not supported by other approaches such as Esteva (2003), Minsky and Ungureanu (2000), Hübner, Sichman and Boissier (2007) and Hübner et al. (2010). There is always a great concern to make sure that agents follow the rules and that agent interaction be more predictable so that it is more feasible to implement MAS.

Our approach allows non-agent software components to interact in the institutional environment by engaging deputy agents to act for them. Also by employing Deputy Agents, it is easier to make changes to the institutional environment interaction protocols – the changes could be made directly and new protocols installed in the Deputy Agents. Of course the new protocols would have to be published to the external users, but using Deputy Agents makes easier both the installation process and the changes located in the part of the application domain related with the role that the Deputy Agent implements.

7.2.1 Social Control

In our approach we define system agents that take care of the organizational structure of the MAS and employ mechanisms to entice external agents to cooperate with the environment, without compromising the unique characteristic of heterogeneous agents, its autonomy. The designer of our kind of open MAS is free to decide the degree of freedom the agents have in the systems by making the system more or less constrained according to the necessity imposed by the application domain.

We do believe that the work from Dellarocas (Dellarocas and Klein, 1999) (Dellarocas, 2000) is a good approach, and one of the problems they confront is how to represent the contracts that regulate the participation of the agents in the society, or the open MAS. We

believe that commitments are a more suitable formalism to define the norms of the system because they are more direct and simple to represent. And a set of commitments that the agents assume with the open MAS can be seen as a contract that the agents assume with the society.

The other point worth mentioning is the identification of the necessity of making explicit the enforcement of the rules of the system; otherwise the agents will not follow them. We use reputation as a concept that should be used to manage enforcement of the rules in the open MAS. As it was explained in Chapters 3 and 4, the commitments the agents have with the society can be broken and their reputations in the system would be updated according to the observation of these commitments by the agents that should follow them. We believe that this allows for a better representation of a truly open MAS where participating agents exercise all the autonomy that they possess. In our approach, we attempt to manage environments composed of truly autonomous agents. After all, this is one of the big aims of the research community, to develop really autonomous agents. Therefore, our aim is to deliver an environment that can fit the requirement of the agent society designer; if he/she needs to have a very constrained environment he/she will be able to do so, and if he/she needs to lower the level of constraints it will be possible to do so, because we offer mechanisms to implement such a versatility.

Our approach is to distribute the management imposed by the centralized middleware in system agents that have predefined management roles and together manage the whole institutional environment. Furthermore, we introduce the freedom of speech to the application agents so that they can have unmonitored interactions regarding any subject they want, making the management of the openness of the environment softer than the usual control imposed by the middlewares, where the agents compulsorily behave as they are told.

Some organisational proxies identified in Organizational Infrastructure implementations are governors in Ameli (Esteva et al., 2004), wrappers in TeamCore (Pynadath and & Tambe, 2003), OrgBox in S-Moise+ (Hübner, Sichman and Boissier, 2006) and controllers in Law-Governed Interaction (Minsky and Ungureanu, 2000). Furthermore, all the communication with the organizational infrastructure is developed through the organizational proxies.

Our approach differs from those, because our agent responsible for the intermediation between the application agents, or external agents, is simply a way for the external agents to

use the interaction protocols available for a determined role. These protocols are represented as Coloured Petri Nets that define in which conversations the external agent can take part in the institutional environment. These agents might have some degree of autonomy if that is desired by the institutional environment designer. Moreover, these agents make available unmonitored interactions which deliver freedom of speech for the external agents participating in the institutional environment. Freedom of speech makes the institutional environment more like human organizations where the human autonomous agents have the choice to follow the rules and can engage in negotiations or conversations that are not predicted by the organizations that they are part of.

The OML in our approach is the formalism we have chosen to represent the conversations and the organizational infrastructure of the institutional environment. It is represented as a set of Coloured Petri Nets (CPN) (Jensen, 1992) that defines the agent roles, both for the application agents, or external agents, and the system agents, which are the agents that manage the institutional environment. Therefore the OI is actually the unification of the CPNs of the agents that are present in the institutional environment, and there is no definition of a middleware that forms the OI. In fact, there is no middleware at all. In our approach the Deputy Agents are a way for the external agents to get in the conversation space of the institutional environment, which is completely distributed in the sense that all the administrative tasks are developed by system agents, and every agent, including the external agents, play their part only in the necessary conversations predicted by the institutional environment designer (see Figure 2.2 in Chapter 2).

It is important to notice that in our approach the Deputy Agent is part of the OI. It is not merely in the frontier with the OI, it actually implements a CPN that is connected with the system agents' CPN forming the OI of the Institutional Environments, differently from other approaches such as Esteva (2003) and Hübner, Sichman and Boissier (2007). Another characteristic that distinguishes the Deputy from the regular proxy agents is that the constraints imposed by the Deputy Agent are lower than those of regular proxy agents. It allows for the use of both predicted and unpredicted conversations through the unmonitored interactions.

The external agents are in the application domain and they actually do not need to be from the same application domain to register with the institutional environment. Furthermore, they can take part in as many institutional environments as they wish or can, as far as they

understands the ontologies that define the application domains of the institutional environments with which it registers.

Our architecture is mainly distributed and the Deputy Agents can actually have some degree of autonomy, if the institutional environment designer decides to supply them with some. Deputy Agents do not force the external agents to engage in only predefined conversations; it is possible for the external agents to meet in the institutional environment and exercise their autonomy by engaging in unpredicted conversations, as far as they are capable of doing so.

Moreover, our OML is not a customized set of symbols or language. We rely on the strong semantics and syntax of the CPN formalism. All the OI is defined as CPNs that together represent a Normative Space for the Institutional Environment.

Regarding the approach of Organizational Artefacts found in Hübner et al. (2010), in our view this is only a matter of allowing external agents to play system agents roles. This is not necessary to implement social order as we have in our model. Moreover, separating the concerns of the system agents from the external agents adds more security and reliability in the operation of the institutional environments.

The operation of the environment is implemented in our approach by the redefinition of reward/sanctioning policies, the way the reputation of the external agents is modified and how it is interpreted by the institutional environment, all this dependent of the application domain. The rules are simple: the external agents can act in the environment and be monitored when they assume compromises in the environment, being punished or rewarded depending on the outcomes of their actions regarding the compromises assumed. Changes in the interpretation of the rules will depend on the implementation of the external agents, their degree of autonomy and how they evolve in time, for example, how they adapt to the rules of the game. By that we mean the norms of the system.

The change of roles in the institutional environment can also be seen as a reorganization of the organizational infrastructure. This means that, when an external agent changes roles, by being promoted to it or by its own choice, it will have a new portion of possible predicted dialogs attached to its CPN that will reorganize the institutional environment as a whole, since it is composed by the merging of all the CPNs that represent external agents and system agents.

Artefacts (such as printers and other devices) play the same role as the proxy agents in the regular organizational infrastructures. They are not autonomous and actually are an interface between the external agents and the environment.

7.2.2. Model Checking and Counts-as Rules

Our approach uses the online monitoring capability of the expectation manager to check a model constantly updated by the introduction of new facts that represent actions that are happening in the institutional environment and consequently creating new states that are added to the history managed by the model checker.

In our approach we represent commitment content and condition through temporal propositions that are checked against a model that represents the state of the institutional environment, regarding the facts that occur in it due to actions of the registered agents. When defining the commitments we reached the point where concepts and facts related with the application domain were getting mixed with concepts and facts from the institutional environment's internal functioning. This was causing a mix of concepts from different levels of abstraction in the definition of the commitments between agents. We opted to use counts-as rules to better accommodate the low-level facts related to the institutional functioning in the commitments, which must take them into consideration to evaluate the more abstract facts related with the application domain.

The mechanism used to implement this approach was the gathering of low-level facts that would translate in an institutional effect necessary for the evaluation of a condition or a content of a commitment against the model defined in the model checker. An example is when the necessity of some system agents be online to execute the checking of a condition that would change the state of a commitment to *active*.

The fact that some agents need to be online for the execution of certain operations in the institutional environment is at a different level of abstraction than the facts that relate with states of the application domain. That fact relates to the state of the institutional environment that is different from the state of the conversations that take place in the institutional environment. The conversations are at a higher level of abstraction because they express concepts and ideas about the application domain. These two kinds of abstraction levels need to be separated implementationally so that the commitments that

express the norms of the institutional environment be better defined and the facts can be more easily checked. That approach brings a separation of concerns necessary for a better modelling and implementation of the commitments that actually need to deal with these different levels of abstraction.

7.2.3 Social Commitments

In our approach the social order is maintained via social commitments, that is, commitments from the external agents with the system agents. This is to represent rules that the external agents should obey for the wellformedness of the institutional environment. It works because the external agent is committing itself to the institutional environment that will behave accordingly with the rules of the open MAS. Moreover, the commitments can regiment specific rules from the application domain such as, for example, an agent can make a bid only once before a certain time slice in an auction. The enforcement in our approach is implemented through the social order mechanisms and the adoption of sanctions by the system agents when commitments are not *fulfilled*.

7.3. Future Work

One way to extend our research would be to experiment with different ways to implement social control using different reputation models in our Institutional Environments. It was out of the scope of this thesis to make this kind of experiment since the main goal was to provide a model to implement an Institutional Environment for open MAS.

The implementation of Collaborative Systems as open MAS and consequently Institutional Environments could be another interesting way to further develop our research. These kinds of systems have very peculiar characteristics: they are distributed, many people participate in them and contribute to their content, and a way to manage the collaboration so that the system does not become less popular or is not a good source of information leads to the modelling of open MAS. We intend to explore more the WikiCrimes system analysing ways to implement better social control in it based on the framework we have developed for Institutional Environments.

More experimentation with the model checking used to change the state of commitment objects can also lead to new research. Particularly with the management of timeouts and deadlines which are not supported by the model checker used in this thesis being these features implemented in the code of the Institution Agent. More experimentation would be done in the sense of adapting these variables to the propositions in the model used by the model checker.

Further research in the manipulation of counts-as rules and institutional fact assertions in the model checker might lead to new developments in the aspects of social control implemented in Institutional Environments. We have used kind of implementational triggers based on events like incoming messages to assert new propositions present in counts-as rules to insert new states in the model present in the model checker. A thorough study of the different levels of abstraction of count-as rules might lead to further development of this feature in our Institutional Environments.

The implementation of the high level CPNs showed in chapter 6 might lead to a higher level of abstraction and interpretation of the Institutional Environments. That will be pursued in future research to further extend the implementation of the Institutional Environment and develop a better graphical user interface for the framework developed and described in this thesis.

More experimentation with the Institutional Environment Framework in different scenarios is another topic to be addressed in the future. The model can be analysed for the implementation of different aspects of open MAS and a methodology for the engineering of Institutional Environment can be developed with more details. Exploitation of the concurrent capabilities offered by the CPNs that implement the Normative Space in different scenarios might lead to fine tuning of the developed framework.

An analysis of the performance and scalability of the method and framework developed will be done as well in future research.

7.4. Contributions

This thesis offers the following contributions:

- Institutional Environments can handle conversations unpredicted at design time allowing more flexibility for the agents to practice their autonomy.
- Social control is implemented in Institutional Environments, providing a way to implement open MAS that are more or less restricted according to the need of the application domain.
- There is flexibility in the definition of the agents' reputation in the open MAS, and therefore the Institutional Environment designer can adjust it to the application domain requirements.
- A framework was developed where the choice of technologies used makes the institutional environment more distributed. That is different from most of the approaches that are followed by other current research, where there is a great use of middlewares to provide a controlled environment for the implementation of organizational infrastructures.
- We have associated Collaborative Systems to open MAS and developed a system that opens a path for further research in that area together with the implementation of social control in that kind of system.

7.4.1. Characteristics of the Developed Framework

- We have built an institutional framework that supports asynchronous agent interaction via FIPA ACL messages. We have agents that implement behaviours in the form of workflows defined as CPNs.
- The framework developed is based on distributed organizational control. System agents are defined for the management of the Institutional Environment.
- Coloured Petri Nets were used to support the modelling and control of concurrent operations.
- We have used the Clojure Functional Language for the annotations of the CPNs that represent the workflows in the Institutional Environment. Clojure as an inscription language proved to be more practical and simple for the manipulation of tokens in the CPNs developed here. The integration of Clojure for the Java Virtual Machine was crucial for the implementation of our system that uses mainly tools based on the Java programming language.

- We have explored counts-as rules to manipulate low level concerns of the Institutional Environment addressing issues that would influence higher level concerns of the agents involved in the Institutional Environment.
- A model checker was used in this thesis for the manipulation of a history of facts, relevant for the change of state of the commitment objects. These commitments represent the social norms in our Institutional Environments.
- We use commitment objects to implement norms in the Institutional Environment. The commitments are managed by system agents and can be used between external agents to stabilish agreements or between system agents and external agents to implement social norms.

7.5. Limitations

The prototype was built with the intent of being as generic as possible and easily used for a variety of application domains, and some issues can be improved in this regard:

- The use of counts-as rules is currently coded in the Institution Agent. There are checks in certain conditions before new states can be inserted in the model from the model checker. We plan to improve the counts-as rules management by the use of rule engine to be invoked prior to the insertion of new states in the model checker.
- We use email addresses to identify uniquely the agents in the prototype implemented. That choice is used also in the existing Web-based WikiCrimes collaborative system. We did not elaborate on managing the reputation of agents based on this identifier but it is a requirement of collaborative systems that there are not many constraints imposed on participants of the system, which could lead to less participation and that is not good in that kind of system. Other ways to identify agents and/or richer models of reputation in the institutional environment might be necessary in other application domains.

Bibliography

Aldewereld, H. (2007) *Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols, PhD Thesis*, Utrecht.

Aldewereld, H., Álvarez-Napagao, S., Dignum, F. and Vázquez-Salceda, J. (2010) 'Making Norms Concrete', Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010), Toronto, 807-814.

Aldewereld, H. and Dignum, V. (2010) 'OperettA: Organizational-Oriented Development Environment', International Workshop on LAnguages, methodologies, and Development tools for multi-agent Systems (LADS).

Aldewereld, H., Dignum, F., Garcia-Camino, A., Noriega, P., Rodriguez-Aguilar, J. and Sierra, C. (2006) 'Operationalization of Norms for Usage in Electronic Institutions', International Joint Conference on Autonomous Agents and Multi-Agent Systems, New York, 223-225.

Alvarez-Napagao, S., Aldewereld, H., Vázquez-Salceda, J. and Dignum, F. (2010) 'Normative Monitoring: Semantics and Implementation', 11th International Workshop on Coordination, Organization, Institutions and Norms in Agent Systems (COIN), Lyon, 1-16.

Arbab, F. (2004) 'Reo: a channel-based coordination model for component composition', *Mathematical Structures in Computer Science*, pp. 329-366.

Arcos, J., Esteva, M., Noriega, P., Rodriguez-Aguilar, J. and Sierra, C. (2005) 'Engineering Open Environments with Electronic Institutions', *Engineeting Applications of Artificial Intelligence*, pp. 191-204.

Artikis, A., Sergot, M. and Pitt, J. (2009) 'Specifying norm-governed computational societies', vol. 1, no. 10, pp. 1-42.

Astefanoaei, L., Dastani, M., Meyer, J. and de Boer, F. (2009) 'On the Semantics and Verication of Normative Multi-Agent Systems', *Journal of Universal Computer Science (J.UCS).*

Austin, J.L. (1962) *How to do things with words*, Oxford University Press.

Baldoni, M., Boella, G., Genovese, V., Grenna, R., Mugnaini, A. and van der Torre, L. (2009) 'A middleware for modeling organizations and roles in JADE', Programming Multi-Agent Systems, First International Workshop, PROMAS, 100-117.

Baldoni, M., Genovese, V., Grenna, R. and van der Torre, L. (2008) 'Adding organizations and roles as primitives to the JADE framework', 3rd International Workshop on Normative Multi-Agent Systems.

Bauer, B., Müller, J.P. and Odell, J. (2001) 'Agent UML: A formalism for specifying multiagent software systems', *International Journal of Software Engineering and Knowledge Engineering*, pp. 207-230.

Bell, M. (2008) *Service-Oriented Modelling (SOA): Service Analysis, Design, and Architecture*, Wiley & Sons.

Bellifemine, F., Poggi, A. and Rimassa, G. (2000) 'Developing Multi-Agent Systems with JADE', International Workshop on Intelligent Agents. Agent Theories Archtectures and Languages., 89-103.

Boella, G. and van der Torre, L. (2008) 'Substantive and procedural norms in normative multiagent systems', *Journal of Applied Logic*, pp. 152-171.

Brafman, M. (1996) 'On partially controlled multi-agent systems', *Journal of Artifitial Intelligence Research.*, pp. 477-507.

Castelfranchi, C. (2000) 'Engineering social order', in Omicini, A., Tolksdorf, R. and Zambonelli, F. (ed.) *Engineering societies in the agent world, first international workshop, ESAW 2000*, Berlin: Springer.

Castelfranchi, C. and Falcone, R. (1998) 'Principles of trust for MAS: Cognitive anatomy, social importance, and quantification', Proceedings of the International Conference on Multi-Agent Systems (ICMAS'98), Paris, 72–79.

Chauhan, D. (1997) *A Java-based Agent Framework for Multiagent Systems Development and Implementation*, Cincinnati.

Colombetti, M., Fornara, N. and Verdicchio, M. (2002) 'The Role of Institutions in Multi-Agent Systems', Workshop on Knowledge Based and Reasoning Agents, 67-75.

Colombetti, M., Fornara, N. and Verdicchio, M. (2004) 'A social approach to communication in multiagent systems', Declarative Agent Languages and Technologies, Melbourne, 191-220.

Colombetti, M. and Verdicchio, M. (2002) 'An analysis of agent speech acts as institutional actions', 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002), Bologna, 1157-1164.

Conte, R. and Paolucci, M. (2002) *Reputation in Artificial Societies. Social Beliefs for Social Order*, Dordrecht: Kluwer Academic Publishers.

Corkill, D.D. and Lesser, V. (1983) 'The use of meta-level control for coordination in a distributed problem solving network', Eighth International Joint Conference on Artifitial Intelligence, Karlsruhe, 748-756.

Cost, R., Chen, Y., Finin, T., Labrou, Y. and Peng, Y. (2000) 'Using Coloured Petri Nets for Conversation Modelling', Issues in Agent Communication, 178-192.

Coutinho, L., Sichman, J.S. and Boissier, O. (2005) 'Modeling organization in MAS: a comparison of models', 1st. Workshop on Software Engineering for Agent-Oriented Systems.

Coutinho, L., Sichman, J.S. and Boissier, O. (2009) 'Modelling Dimensions for Agent Organizations', in Dignum, V. (ed.) *Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, IGI Global.

Cranefield, S. (2006) 'A rule language for modelling and monitoring social expectations in multiagent systems', Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems, 246–258.

Cranefield, S. and Purvis, M. (2002) 'A UML Profile and Mapping for the Generation of Ontology-Specific Content Languages', *The Knowledge Engeneering Review*, vol. 17, no. 1, pp. 21-39.

Cranefield, S., Purvis, M., Nowostawski, M. and Hwang, P. (2002) 'Ontologies for Interaction Protocols', Ontologies for Agents: Theory and Experiences, 1-18.

Cranefield, S. and Winikoff, M. (2009) 'Verifying social expectations by model checking truncated paths', Coordination, Organizations, Institutions and Norms in Agent Systems IV (LNCS), 204-219.

Cranefield, S. and Winikoff, M. (2011) 'Verifying social expectations by model checking truncated paths', *Journal of Logic and Computation*, pp. 1217-1256.

Criado, N., Argente, E. and Botti, V. (2011) 'THOMAS: An agent platform for supporting normative multi-agent systems', *Journal of Logic and Computation Advance Access*, 7 September.

Dastani, M. (2009) 'Normative multi-agent organizations', in *Engineering Societies in the Agents World X*, 5881st edition, Springer.

Dastani, M., Grossi, D., Meyer, J. and Tinnemeier, N. (2008) 'Normative multi-agent programs and their logics', Workshop on Knowledge Representation for Agents and Multi-Agent Systems, 16-31.

De Oliveira, M., Cranefield, S. and Purvis, M. (2007) 'Normative Spaces in Institutional Environments by the means of Commitments, Reputation and Coloured Petri Nets', International Workshop of Agent Oriented Software Engineering, Honolulu.

De Oliveira, M., Furtado, V., Cranefield, V. and Purvis, M. (2009) 'Institutional Environments and Open Collaborative Systems', 5th Workshop on Software Engineering for Agent-Oriented Systems, Fortaleza, 13-24.

De Oliveira, M., Furtado, V., Purvis, M. and Cranefield, S. (2008) 'Open Collaborative Systems as Institutions of Agents', Proceedings of the International Conference on Inteligent Agent Technology, Sydney, 100-103.

De Oliveira, M. and Purvis, M. (2008) 'Aspects of openness in multi-agent systems: coordinating the autonomy in agent societies', in Krol, D. and Nguyen, N.T. (ed.) *Intelligent Integration in Distributed Knowledge Management*, New York.

De Oliveira, M., Purvis, M., Cranefield, S. and Nowostawski, M. (2004) 'A Distributed Model for Institutions in Open Multi-Agent Systems', in Negoita, R., Howlett, J. and Jain, L.C. (ed.) *Knowledge-Based Intelligent Information and Engineering Systems*, Springer-Verlag.

De Oliveira, M., Purvis, M., Cranefield, S. and Nowostawski, M. (2004) 'Institutions and commitments in open multi-agent systems', IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2004), 500-503.

De Oliveira, M., Purvis, M., Cranefield, S. and Nowostawski, M. (2005) 'The Role of Ontologies when Modelling Open Multi-Agent Systems as Institutions', in Katarzyniak, R.P. (ed.) *Ontologies and Soft Methods in Knoledge Management*, Adelide: Advanced Knowledge International.

Decker, K. (1996) 'TAEMS: A framework for environment centered analysis & design of coordination mechanisms', in O'Hare, G. and Jennings, N. (ed.) *Foundations of Distributed Artificial Intelligence*.

Dellarocas, C. (2000) 'Contractual Agent Societies: Negotiated Shared Context and Social Control in Open Multi-Agent Systems', Workshop on Norms and Institutions in Multi-Agent Systems, Barcelona, 113-134.

Dellarocas, C. and Klein, M. (1999) 'Civil agent societies: Tools for inventing open agent-mediated electronic marketplaces', ACM Conference on Electronic Commerce (EC-99), 24-39.

DeLoach, S. (2002) 'Modeling Organizational Rules in the Multiagent Systems Engineering Methodology', Proceedings of the 15th Canadian Conference on Artificial Intelligence, Calgary, 1-15.

Dignum, F. (2002) 'Abstract norms and electronic institutions', International Workshop on Regulated Agent-Based Social Systems: Theories and Applications (RASTA), Netherlands.

Dignum, V. (2004) *A model for organizational interaction: based on agents, founded in logic*, PhD Thesis edition.

Dignum, V., Aldewereld, H. and Dignum, F. (2011) 'On the Engineering of Multi Agent Organizations', 12th Workshop on Agent Oriented Software Engineering, 53-65.

Dignum, F. and Greaves, M. (2000) 'Issues in agent communication: An introduction.', in Dingnum, F. and Greaves, M. (ed.) *Issues in Agent Communication*, Springer-Verlag.

Dragone, L. (2005) *Hybrid logics model checker*.

Dybvig, R. (2003) *The Scheme Programming Language*, MIT Press.

Elio, R. and Haddadi, A. (1999) 'On Abstract Task Models and Conversation Policies', Workshop on Specifying and Implementing Conversation Policies, 89-98.

Esteva, M. (2003) *Electronic Institutions: from specification to development*, Bellaterra: Institut d'Investigació en Intelligència Artificial - IIIA.

Esteva, M., de la Cruz, D. and Sierra, C. (2002) 'ISLANDER: An electronic institutions editor', Proceedings of the first international joint conference on autonomous agents and multiAgent systems (AAMAS 2002), 1045–1052.

Esteva, M., Rodrígues-Aguilar, J.A., Sierra, C., Arcos, J.L. and Garcia, P. (2001) 'On the formal specification of electronic institutions', Agent-mediated Electronic Commerce: The European AgentLink Perspective, 126-147.

Esteva, M., Rodríguez-Aguilar, J.A., Rosell, B. and Arcos, J.L. (2004) 'AMELI: An agent-based middleware for electronic institutions', Proceedings of the third international joint conference on autonomous agents and multi-agent systems (AAMAS 2004), New York, 236–243.

Ferber, J., Gutknecht, O. and Michel, F. (2004) 'From agents to organizations: an organizational view of multi-agent systems', in Giorgini, P., Müller, J.P. and Odell, J. (ed.) *Agent Oriented Software Engineering IV*, 2935th edition, Springer.

Ferber, J. and Gutknetch, O. (1998) 'A metamodel for the analysis of organizations in multiagents systems', Third International Conference on Multiagent Systems (ICMAS-98), 128-135.

Ferber, J., Michel, F. and Baez, J. (2005) 'AGRE: Integrating environments with organizations', Environments for Multi-Agent Systems, E4MAS, 48–56.

Finin, T., Labrou, Y. and Mayfield, J. (1997) 'KQML as an Agent Communication Language', *Software Agents*, pp. 291-316.

FIPA (1997) *FIPA 97 Specification Part 2: Agent Communication Language*.

FIPA (2000) *FIPA specifications (http://www.fipa.org/repository/aclspecs.php3)*, Available: http://www.fipa.org/specifications/index.html.

FIPA (2001) *FIPA Agent Management Specification (SC00023K)*, Geneva.

FIPA (2002) *FIPA ACL Message Structure Specification (SC00061)*.

FIPA (2002) *FIPA Agent Transport Sercice Specification (SC00067)*, Geneva.

Fornara, N. and Colombetti, M. (2002) 'Operational specification of a commitment-based agent communication language', First international joint conference on Autonomous agents and multiagent systems, 536-542.

Fornara, N., Viganò, F., Verdicchio, M. and Colombetti, M. (2008) 'Artificial institutions: a model of institutional reality for open multiagent systems', in *Artificial Intelligence and Law*.

Fouss, J.D. and Chang, K.H. (2000) 'Classifying Groupware', Proceedings of the 38th ACM Southeast Conference, Clemson, 117–124.

Furtado, V., Ayres, L., de Oliveira, M., Vasconcelos, E., Caminha, C., D'Orleans, J. and Belchior, M. (2010) 'Collective intelligence in law enforcement – The WikiCrimes system', *Information Sciences - Informatics and Computer Science Intelligent System Applications*, vol. 180, no. 1, January, pp. 4-17.

Gaertner, D., Garcia-Camino, A., Noriega, P., Rodriguez-Aguilar, J.-A. and Vasconcelos, W. (2007) 'Distributed norm management in regulated multiagent systems', 6th International Joint Conference on Autonomous Agents and Multiagent Systems, 624-631.

Galan, A.K. (2000) *JiVE: JAFMAS Integrated Visual Environment*, Cincinnati.

García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C. and Vasconcelos, V. (2009) 'Constraint Rule-based Programming of Norms for Electronic Institutions', Autonomous Agents and Multi-Agent Systems, 186-217.

Gasser, L., Braganza, C. and Herman, N. (1987) 'MACE: A flexible test-bed for distributed AI research', in *Distributed Artificial Intelligence*, Pitman Publishers.

Gelernter, D. and Zuck, L.D. (1997) 'On what linda is: Formal description of linda as a reactive system', in Garlan, D. and Le Métayer, D. (ed.) *Coordination Languages and Models*, Springer.

Genesereth, M.R. and Ketchpel, S.P. (1994) 'Software Agents', *Communications of the ACM*, pp. 48-53.

Giunchiglia, F., Mylopoulos, J. and Perini, A. (2002) 'The tropos software development methodology: Process, models and diagrams', Workshop on Agent Oriented Software Engineering (AOSE-2002), 63-74.

Greaves, M., Holmback, H. and Bradshaw, J. (2000) 'What is a conversation policy?', *Issues in Agent Communication*, pp. 118-131.

Grossi, D., Aldereweld, H. and Dignum, F. (2007) 'Ubi Lex, Ibi Poena: Designing norm enforcement in e-institutions', Coordination, organizations, institutions, and norms in agent systems II, 101–114.

Grossi, D., Aldereweld, H., Vázquez-Salceda, J. and Dignum, F. (2006) 'Ontological aspects of the implementation of norms in agent-based electronic institutions', in *Computational and Mathematical Organization Theory*, Springer.

Grudin, J. and Poltrock, S. (1991) 'Computer supported cooperative work and groupware', Tutorial presented at CM SIGCHI Conference on Human Factors in Computing Systems, New Orleans, 269-320.

Gutknecht, O. and Ferber, J. (2000) 'MadKit: a generic multi-agent platform', Fourth international conference on Autonomous agents, 78-79.

Gutnik, G. and Kaminka, G.A. (2006) 'Representing Conversations for Scalable Overhearing', *Journal of Artificial Intelligence Research*, pp. 349-387.

Halloway, S. (2009) *Programming Clojure*, Texas: The Pragmatic Bookshelf.

Hanachi, C. and Sibertin-Blanc, C. (2004) 'Protocol Moderators as Active Middle-Agents in Multi-Agent Systems', Autonomous Agents and Multi-Agent Systems, 131-164.

Horling, B. and Lesser, V. (2005) 'A Survey of Multi-Agent Organizational Paradigms', *The Knowledge Engineering Review*, vol. 4, no. 19, pp. 281-316.

Howden, N., Romquist, R., Hodgson, A. and Lucas, A. (2001) 'Jack intelligent agents - summary of an agent infrastructure', Fifth International Conference on Autonomous Agents.

Howe, J. (2008) *Crowdsourcing: Why the Power of the Crowd Is Driving the Future of Business*.

Hübner, J., Boissier, O., Kitio, R. and Ricci, A. (2010) 'Instrumenting multi-agent organisations with organisational artifacts and agents “Giving the organisational power back to the agents”', Autonomous Agents and Multi-Agent Systems Conference (AAMAS), 369-400.

Hübner, J.F., Sichman, J. and Boissier, O. (2002) 'Moise+: towards a structural, functional, and deontic model for MAS organization', AAMAS, 501-502.

Hübner, J., Sichman, J. and Boissier, O. (2006) 'S-MOISE+: A middleware for developing organised multi-agent systems', Coordination, organizations, institutions, and norms in multi-agent systems, 64–78.

Hübner, J., Sichman, J. and Boissier, O. (2007) 'Developing organised multi-agent systems using the MOISE+ model: Programming issues at the system and agent levels', *International Journal of Agent-Oriented Software Engineering*, p. 370–395.

Ierusalimschy, R., De Figueiredo, L. and Celes, W. (2005) 'The Implementation of Lua 5.0', *Journal of Universal Computer Science*, pp. 1159-1176.

Iglesias, C.A., Garijo, M. and Gonzalez, J.C. (1999) 'A survey of agent oriented methodologies', Intelligent Agents V, 317-330.

Jennings, N.R. (2000) 'On Agent based software engineering', *Artificial Intelligence*, pp. 277-296.

Jennings, N.R., Sycara, K. and Wooldridge, M. (1998) 'A roadmap of agent research and development', Autonomous Agents amd Multiagent Systems, 275-306.

Jensen, K. (1992) 'Coloured Petri Nets - Basic Concepts, Analysis, Methods and Pratical Use', in *EATCS Monographs on Theoretical Computer Science*.

Juan, T., Pierce, A. and Sterling, L. (2002) 'Roadmap: Extending the Gaia methodology for complex open systems', First International Joint Conference o Autonomous Agents and Multiagent Systems (AAMAS 2002), 3-10.

Juneau, J., Baker, J., Ng, V., Soto, S. and Wierzbicki, F. (2010) *The Definitive Guide to Jython: Python for the Java™ Platform*, APress.

Kahn, T. (2008) *Boletim de Ocorrência – prover para poder prever*, [Online], Available: http://www.forumseguranca.org.br/artigos/ [September 2008].

Koning, J.L., Francois, G. and Demazeau, Y. (1998) 'Formalization and pre-validation for interaction protocols in multiagent systems.', 13th European Conference on Artificial Intelligence, 298-302.

Labrou, Y., Finin, T. and Peng, Y. (1999) 'Agent communication languages: The current landscape.', *IEEE Inteligent Systems*, pp. 45-52.

Lee, T., Hendler, J. and Lassila, O. (2001) 'The Semantic Web', *Scientific American*, vol. 5, pp. 28-37.

Lesser, V. (2004) 'Evolution of the GPGP/TAEMS domain independent coordination framework', *Autonomous Agents and Multi-Agent Systems*, vol. 9, no. 1, pp. 87-143.

Lopez y Lopez, F., Luck, M. and d'Inverno, M. (2006) 'A normative framework for agent-based systems', *Comp. & Math. Org. Theory*, vol. 12, no. 2-3, pp. 227-250.

Markey, N. and Schnoebelen, P. (2003) 'Model checking a path', CONCUR 2003 – Concurrency Theory, 251–265.

Martin, F. and Kendall, S. (2000) *UML distilled: a brief guide to the standard object modeling language*, 2nd edition, Boston: Addison-Wesley Longman Publishing Co.

Martìn, F.J., Plaza, E. and Rodrìguez-Aguilar, J.A. (2000) 'As infrastructure for agent-based systems: An untegrated approach', *International Journal of Intelligent Systems*, pp. 217-240.

Mazoui, H., Fallah, A.E. and Haddad, S. (2002) 'Open protocol design for complex interactions in multi-agent systems', First International Joint Conference on Autonomous Agents and Multiagent Systems, 517-526.

McKnight, D.H. and Chervany, N.L. (2001) 'Trust and distrust definitions: One bite at a time', Proceedings of theWorkshop on "Deception, Fraud and Trust in Agent Societies", Montreal, 27–54.

Melaye, D. and Demazeau, Y. (2005) 'Bayesian dynamic trust model', Proceedings of the Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'05), Budapest, 480–489.

Milner, R. (1990) *The Definition of Standard ML*, MIT Press.

Minsky, N. and Ungureanu, V. (2000) 'Law-Governed Interaction: a Coordination and Control Mechanism for Heterogeneous Distributed Systems', *ACM Transactions on Software Engeneering and Methodology*, vol. 9, no. 3, pp. 273-305.

Muller, G. and Vercouter, L. (2010) 'L.I.A.R.: Achieving Social Control in Open and Decentralised Multi-Agent Systems', *Applied Artificial Intelligence*, pp. 723-768.

Nieves, J.C., Padget, J., Vasconcelos, W., Staikopoulos, A., Cliffe, O., Dignum, F., Vazquez-Salceda, J., Clarke, S. and Reed, C. (2010) 'Coordination, Organisation and Model Driven Approaches for Dynamic, Flexible, Robust Software and Services Engineering', in Schahram, D. and Li, F. *Service Engineering*, Springer.

Nodine, M.H. and Unruh, A. (1999) 'Constructing robust conversation policies in dynamic agent communities', Workshop on Specifying and Implementing Conversation Policies, 205-219.

Noriega, P. (1997) *Agent-Mediated Auctions: The Fishmarket Metaphor*.

Noriega, P. (2006) 'Fencing the Open Fields: EMpirical Concerns on Electronic Institutions', Coordination, Organizations, Institutions and Norms in Multi-Agent Systems, 81-98.

North, D. (1990) *Institutions, Institutional Change and Economics Performance*, Cambridge U. P.

Nowostawski, M. (2009) *JFern Manual*, 4th edition, Dunedin.

Nowostawski, M., De Oliveira, M., Purvis, M. and Cranefield, S. (2006) 'Institutions in the Opal Multi-Agent System', *Journal of Intelligent and Fuzzy Systems*, pp. 192-207.

Nowostawski, M., Purvis, M. and Cranefield, S. (2002) 'OPAL - A Multi-level Infractructure for Agent-Oriented Development', International Joint COnference on Autonomous Agents and Multi-Agent Systems, 88-89.

Odell, J., Parunak, H. and Bauer, B. (2000) 'Extending UML for Agents', Agent-Oriented Information Systems Workshop, 3-17.

OMG *OMG: Model Driven Architecture*, [Online], Available: http://www.omg.org/mda/ [30 Jan 2013].

O'Reilly, T. (2007) *What is Web 2.0: design patterns and business models for the next generation of software, Communications and Strategies*.

Padgham, L. and Winikoff, M. (2002) 'Prometheus: A methodology for developing intelligent agents', Workshop on Agent Oriented Softaware Engeneering (AOSE), 135-145.

Parunak, H.V.D. and Odell, J. (2002) 'Representing social structures in UML', Agent-Oriented Software Engineering Workshop II, 1-16.

Paschke, A., Dietrich, J. and Kuhla, K. (2005) 'A logic based sla management framework', 4th Semantic Web Conference (ISWC), 68-83.

Pattison, H.E., Corkill, D.D. and Lesser, V.R. (1987) 'Instantiating Descriptios of Oraganizational Structures', Distributed Artifitial Intelligence, 59-96.

Poslad, S. (2009) *Ubiquitous Computing Smart Devices, Smart Environments and Smart Interaction*, Wiley.

Purvis, M., Nowostawski, M. and Cranefield, S. (2002) 'A Multi-level Infrastructure for Agent-Oriented Development', AAMAS, 88-89.

Purvis, M., Nowostawski, M., De Oliveira, M. and Cranefield, S. (2003) 'Multi-agent Interaction Protocols for E-Business', IEEE/WIC International Conference on Intelligent Agent Technology (IAT), Los Alamitos, 310-324.

Purvis, M.K., Savarimuthu, S., De Oliveira, M. and Purvis, M. (2006) 'Mechanisms for Cooperative Behaviour in Agent Institutions', IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT), Chuc Hay, 121-124.

Pynadath, D.V. and & Tambe, M. (2003) 'An automated teamwork infrastructure for heterogeneous software agents and humans', Autonomous Agents and Multi-Agent Systems, 71–100.

Rao, A.S. and Georgeff, M.P. (1995) 'BDI Agents: from theory to practice', First International Conference on Multiagent Systems, 312-319.

Reisig, W. (1985) *Petri Nets. An Introduction*, Springer.

Ricci, A. and Ominici, A. (2003) 'Supporting Coordination in Open Computational Systems with Tucson', WETICE, 365-370.

Ricci, A., Viroli, M. and Omicini, A. (2007) 'Give agents their artifacts: the a&a approach for engineering working environments in mas', 6th International Joint Conference on Autonomous Agents and Multiagent Systems, Honolulu, 1-3.

Rodríguez-Aguilar, J.A. (2001) *On the Design and Construction of Agent-mediated Electronic Institutions*.

Rymaszewski, M. (2007) *Second life: the official guide*, Indianapolis: Wiley Publishing.

Sabater, J. (2002) *Trust and Reputation for Agent Societies*, PhD Thesis edition, Barcelona.

Searle, J.R. (1969) *Speech acts*, Cambridge University Press.

Searle, J.R. (1995) *The Construction of Social Reality*, London: The Penguin Press.

Sibertin-Blanc, C. (2001) 'Cooperative objects: Principle, use and implementation', Concurrent Object-Oriented Programming and Petri Nets, 216-246.

Sibertin-Blanc, C., Hamachi, C. and Cardoso, J. (2000) 'Communication Protocols as a first-class components of multiagent systems', Fourth Internacional Conference on Multi-agent Systems, 437-438.

Silva, V., Choren, R. and Lucena, C. (2004) 'A UML based approach for modeling and implementing multi-agent systems', Third Inter. Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS, 914-921.

Singh, M. (1999) 'An Ontology for Commitments in Multi-Agent Systems', *Artificial Intelligence and Law*, pp. 97-113.

Tambe, M., Adibi, J., Alonaizan, Y., Erdem, A., Kaminka, G., Marsella, S. and Muslea, I. (1999) 'Building agent teams using an explicit teamwork model and learning', *Artificial Intelligence*, vol. 110, p. 215–240.

Tinnemeier, N., Dastani, M. and Meyer, J. (2008) 'Orwell's nightmare for agents? programming multi-agent organisations', ProMAS'08, 56-71.

Tinnemeier, N., Dastani, M. and Meyer, J. (2009) 'Roles and norms for programming agent organizations', The 8th International Conference on Autonomous Agents and Multiagent Systems, 121-128.

van Riemsdijk, M.B., Hindriks, K.V. and Jonker, C.M. (2009) 'Programming organization-aware agents: A research agenda', Tenth International Workshop on Engineering Societies in the Agents' World (ESAW), 98-112.

Vàzquez-Scaleda, J. and Dignum, F. (2003) 'Modelling electronic organizations', Thrird International/Central and Eastern European COnference on Multi-Agent Systems, 1070-1070.

Verdicchio, M. and Colombetti, M. (2009) 'Communication Languages for Multiagent Systems', *Computational Intelligence*, May, pp. 136-159.

Werner, E. (1987) 'A united theory of communication and social structure', in *Distributed Artifitial Intelligence*, Pitman Publishers.

Wooldridge, M. and Ciancarini, P. (2001) 'Agent-oriented software engeneering: The state of the art', Agent-oriented Software Engeneering, 285-312.

Wooldridge, M. and Jennings, N. (1999) 'Software engeneering with agents: pitfalls and pratfalls', *IEEE Internet Computing*, pp. 20-27.

Wooldridge, M., Jennings, N. and Kinny, D. (2000) 'The Gaia Methodology for Agent-Oriented Analysis and Design', Autonomous Agents and Multi-Agent Systems, 285-312.

Zambonelli, F., Jennings, N. and Wooldridge, M. (2001) 'Organisational abstractions for the analysis and design of multi-agent systems', Agent-Oriented Software Engineering, LNCS, 407-422.

Zambonelli, F., Jennings, N. and Wooldridge, M. (2003) 'Developing Multi-Agent Systems: The Gaia Methodology', *ACM Transactions on Software Engineering and Methodology*, vol. 12, no. 3, pp. 317-370.

Appendix A

Note that the inscriptions are written in Clojure language.

1. Annotations of the Institution Agent Workflow Coloured Petri Net (Figure 5.2)

1.1. Transition `Set_Environment`

- From Place `Initialization` to Transition `Set_Environment`:

Guard: `(instance? nzdis.agent.OpalAgent i)`

Expression: `var("i");`

- Transition `Set_Environment`:

Action: `(def agent_owner i)`

1.2. Transition `Registration`

- Message content example: `(registration :role RegisteredUser :eternal_email 'teste@wikicrimes.com')`

- Input Arc from Place `Ext_Agent_Inc_Msg` to Transition `Registration`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "register" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Registration` to Place `Reputation_Agent_Out_Msg`:

Expression: `msgReputation`

- Output Arc from Transition `Registration` to Place `MonitorAgent_Out_Msg`:

Expression: `msgMonitor`

- Output Arc from Transition `Registration` to Place `Ext_Agent_Out_Msg`:

Expression: `msgExternal`

- Transition `Registration`:

Action:

```
(def registration (vec (re-seq #"\w+" (. x get "content"))))

(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def external_email (get (get quoted 0) 1))

(def new_deputy (. agent_owner getNewDeputy (get registration
2) external_email)

(def msgReputation (new nzdis.agent.message.Message "REQUEST"
( apply str "(create_file :external_email '" external_email
"' :deputy '" (. new_deputy getAgentNameString) "')")))

(. msgReputation setSender (. agent_owner getAgentName))

(. msgReputation setReceiver
("ReputationAgent@OtagoPlatform"))

(def msgMonitor (new nzdis.agent.message.Message "REQUEST" (
apply str "(make_commitment :creditor
InstitutionAgent@OtagoPlatform :debitor '" (. new_deputy
getAgentNameString) "' :condition 'and ("(. new_deputy
getAgentNameString) "_submited_report " (. new_deputy
getAgentNameString) "_Report_has_status_accepted)' :content
'not(or (" (. new_deputy getAgentNameString)
"_Negative_confirmation_on_report_is_made))')")))

(. msgMonitor setSender (. agent_owner getAgentName))

(. msgMonitor setReceiver ("MonitorAgent@OtagoPlatform"))

(def msgExternal (new nzdis.agent.message.Message "INFORM" (
apply str "(register :status ok :deputy '" (. new_deputy
getAgentNameString)"')")))

(. msgExternal setSender (. agent_owner getAgentName))

(. msgExternal setReceiver (. x getSender))
```

1.3. Transition `ChangeRole`

- Message content example: `(change_role :new_role CertifierEntity :external_email 'teste@wikicrimes.com')`

- Input Arc from Place `Deputy_Agent_Inc_Msg` to Transition `ChangeRole`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "change_role" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `ChangeRole` to Place `Reputation_Agent_Out_Msg`:

Expression: `msgReputationChangeRole`

- Output Arc from Transition `ChangeRole` to Place `MonitorAgent_Out_Msg`:

Expression: `msgMonitorChangeRole`

- Output Arc from Transition `ChangeRole` to Place `Ext_Agent_Out_Msg`:

Expression: `msgExternalChangeRole`

- Transition `ChangeRole`:

Action:

```
(def change_role (vec (re-seq #"\w+" (. x get "content"))))

(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def external_email (get (get quoted 0) 1))

(if (. agent_owner checkRoleChangePossibility (get
change_role 2) external_email)

(

(def new_deputy_change_role (. agent_owner getNewDeputy (get
change_role 2) external_email)

(def msgReputationChangeRole (new nzdis.agent.message.Message
"REQUEST" ( apply str "(update_change_role :external_email '"
```

```
external_email "' :deputy '" (. new_deputy_change_role
getAgentNameString) "')")))

(. msgReputationChangeRole setSender (. agent_owner
getAgentName))

(. msgReputationChangeRole setReceiver
("ReputationAgent@OtagoPlatform"))

(def msgMonitorChangeRole (new nzdis.agent.message.Message
"REQUEST" ( apply str "(assign_commitments
:current_debtor_deputy '" (. x getSender) "'
:new_debitor_deputy '" (. new_deputy_change_role
getAgentNameString) "')")))

(. msgMonitorChangeRole setSender (. agent_owner
getAgentName))

(. msgMonitorChangeRole setReceiver
("MonitorAgent@OtagoPlatform"))

(def msgExternalChangeRole (new nzdis.agent.message.Message
"INFORM" ( apply str "(change_role :status ok :deputy '" (.
new_deputy_change_role getAgentNameString) "')")))

(. msgExternalChangeRole setSender (. agent_owner
getAgentName))

(. msgExternalChangeRole setReceiver (. agent_owner
getExternalReference (. x getSender)))

)

(

(def msgMonitorChangeRole nil)

(def msgReputationChangeRole nil)

(def msgExternalChangeRole (new nzdis.agent.message.Message
"INFORM" "(change_role :status fail)"))

(. msgExternalChangeRole setSender (. agent_owner
getAgentName))

(. msgExternalChangeRole setReceiver (. agent_owner
getExternalReference (. x getSender)))

)
```

1.4. Transition `Unregistration_Request`

- Message content example: `(unregistration_deputy)`

- Input Arc from Place `Deputy_Agent_Inc_Msg` to Transition `Unregistration_Request`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "unregistration_deputy" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

Output Arc from Transition `Unregistration_Request` to Place `MonitorAgent_Out_Msg`:

Expression: `msgMonitorUnregistrationRequest`

- Transition `Unregistration_Request`:

Action:

```
(def msgMonitorUnregistrationRequest (new
nzdis.agent.message.Message "REQUEST" ( apply str
"(unregistration_request :deputy '" (. x getSender) "')")))

(. msgMonitorUnregistrationRequest setSender (. agent_owner
getAgentName))

(. msgMonitorUnregistrationRequest setReceiver
("MonitorAgent@OtagoPlatform"))
```

1.5. Transition `Unregistration_Request_Processed`

- Message content example: `(unregistration_processed :deputy 'RegisteredUser01@OtagoPlatform')`

- Input Arc from Place `MonitorAgent_Inc_Msg` to Transition `Unregistration_Request_Processed`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "unregistration_processed" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Unregistration_Request_Processed` to Place `Ext_Agent_Out_Msg`:

Expression: `msgExternalUnregistrationRequestProcessed`

- Transition `Unregistration_Request_Processed`:

Action:

```
(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def deputy (get (get quoted 0) 1))

(def msgExternalUnregistrationRequestProcessed (new
nzdis.agent.message.Message "INFORM"
"(unregistration_request_processed :status ok)"))

(. msgExternalUnregistrationRequestProcessed setSender (.
agent_owner getAgentName))

(. msgExternalUnregistrationRequestProcessed setReceiver (.
agent_owner getExternalReference deputy))

)
```

1.6. Transition `Query_ModelChecker`

- Message content example: `(evaluate_current_state :commitments (Commitment01 Commitment02 Commitment03))`

- Input Arc from Place `MonitorAgent_Inc_Msg` to Transition `Query_ModelCecker`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "evaluate_current_state" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Query_ModelChecker` to Place `MonitorAgent_Out_Msg`:

Expression: `msgMonitorQueryResponce`

- Transition `Query_ModelChecker`:

Action:

```
(def query_content (vec (re-seq #"\w+" (. x get "content"))))

(def commitments (rest (rest query_content)))

(def result_evaluation (for [c commitments] (. agent_owner
evaluateCommitmentContent c)))
```

```
(def query_responce (interleave commitments
result_evaluation))

(def msgMonitorQueryResponce (new nzdis.agent.message.Message
"INFORM" ( apply str "(query_responce :commitments (" (apply
str query_responce) ")")))

(. msgMonitorQueryResponce setSender (. agent_owner
getAgentName))

(. msgMonitorQueryResponce setReceiver
("MonitorAgent@OtagoPlatform"))
```

1.7. Transition `Update_ModelChecker_by_Deputy`

- Message content example: `(update_modelchecker :truthvalue true :fact 'fact1')`

- Variable `result_update` example: `"Commitment01 active Commitment02 fulfilled Commitment03 violated"`

- Input Arc from Place `Deputy_Agent_Inc_Msg` to Transition `Update_ModelChecker_by_Deputy`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "update_modelchecker" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Update_ModelChecker_by_Deputy` to Place `MonitorAgent_Out_Msg`:

Expression: `msgMonitorUpdateByDeputy`

- Transition `Update_ModelChecker_by_Deputy`:

Action:

```
(def update_content (vec (re-seq #"\w+" (. x get
"content"))))

(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def incoming_fact (get (get quoted 0) 1))

(def fact (apply str ((. x getSender) "_" incoming_fact )))
```

```
(def result_update (. agent_owner updateModelInsertFact fact
(get update_content 2)))

(def msgMonitorUpdateByDeputy (new
nzdis.agent.message.Message "INFORM" ( apply str
"(set_commitment :updates (" result_upate ")")))

(. msgMonitorUpdateByDeputy setSender (. agent_owner
getAgentName))

(. msgMonitorUpdateByDeputy setReceiver
"MonitorAgent@OtagoPlatform")
```

1.8. Transition `Update_ModelChecker_by_Monitor`

- Message content example: `(update_modelchecker_insert :commitment Commitment01 :condition_formula 'formula1' :content_formula 'formula2')`
- Message content example: `(update_modelchecker_assign :old_debtor 'RegisteredUser01@OtagoPlatform' :new_debtor 'RegisteredUser01@OtagoPlatform' :commitments (Commitment01 Commitment02 Commitments03))`

- Input Arc from Place `MonitorAgent_Inc_Msg` to Transition `Update_ModelChecker_by_Monitor`:

Guard: `(and (instance? nzdis.agent.message.Message x) (or (if (= "update_modelchecker_insert" (first (re-seq #"\w+" (. x get "content"))) true false) (if (= "update_modelchecker_cancel" (first (re-seq #"\w+" (. x get "content"))) true false) (if (= "update_modelchecker_release" (first (re-seq #"\w+" (. x get "content"))) true false) (if (= "update_modelchecker_discharge" (first (re-seq #"\w+" (. x get "content"))) true false) (if (= "update_modelchecker_assign" (first (re-seq #"\w+" (. x get "content"))) true false)))`

Expression: `var("x");`

- Transition `Update_ModelChecker_by_Monitor`:

Action:

```
(def update_content (vec (re-seq #"\w+" (. x get
"content"))))
```

```
(if (= (get update_content 0) "update_modelchecker_assign") (

(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def old_debtor (get (get quoted 0) 1))

(def new_debtor (get (get quoted 0) 1))

(def commitmentIDs (rest(rest(rest(rest(rest(rest(rest(rest
update_content)))))))))

(def helper (repeat " " (count commitmentIDs)))

(. agent_owner updateModelAssignCommitments old_debtor
new_debtor (apply str (interleave commitmentIDs helper)))

)

(

(def formulas (vec (re-sec #"'(.*?)'" (. X get "content"))))

(def commitment_condition_name (apply str "Condition_" (get
update_content 2)))

(def condition_formula (get (get formulas 0) 1))

(def commitment_content_name (apply str "Content_" (get
update_content 2)))

(def content_formula (get (get formulas 1) 1))

)

(if (= (get update_content 0) "update_modelchecker_insert")
(. agent_owner updateModelInsertCommitment
commitment_condition_name condition_formula
commitment_content_name content_formula))

(if (= (get update_content 0) "update_modelchecker_cancel")
(. agent_owner updateModelCancelCommitment
commitment_condition_name condition_formula
commitment_content_name content_formula))

(if (= (get update_content 0) "update_modelchecker_release")
(. agent_owner updateModelReleaseCommitment
commitment_condition_name condition_formula
commitment_content_name content_formula))

(if (= (get update_content 0)
"update_modelchecker_discharge") (. agent_owner
```

```
updateModelDischargeCommitment commitment_condition_name
condition_formula commitment_content_name content_formula))
```

1.9. Transition `Observation_Area_Alert`

- Message content example: `(observation_area_alert (observation_area :observation_area_id 3 :crime_id 1 :agent_owner_email 'RegisteredUser01@email.ac.nz'))`

- Input Arc from Place `DBAdmAgent_Inc_Msg` to Transition `Observation_Area_Alert`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "observation_area_alert" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Observation_Area_Alert` to Place `DeputyAgent_Out_Msg`:

Expression: `msgObervationAreaAlert`

- Transition `msgObservation_Area_Alert`:

Action:

```
(def alert_content (vec (re-seq #"\w+" (. x get "content"))))

(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def agent_owner_email (get (get quoted 0) 1))


(def msgObservation_Area_Alert (new
nzdis.agent.message.Message "INFORM" ( apply str
"(observation_area_alert :observation_area_id " (get
alert_content 3) ":crime_id " (get alert_content 5) ")"))

(.msgObservation_Area_Alert setSender (. agent_owner
getAgentName))

(.msgObservation_Area_Alert setReceiver (. agent_owner
getDeputyReference agent_owner_email))

)
```

1.10. Transition SendInvitation

- Message content example: (crime_confirmation_invitation :invitation_id 1 :invited_user_email 'xyz125@gmail.com')

- Input Arc from Place DBAdmAgent_Inc_Msg to Transition SendInvitation:

Guard: (and (instance? nzdis.agent.message.Message x)(if (= "crime_confirmation_invitation" (first (re-seq #"\w+" (. x get "content"))) true false))

Expression: var("x");

- Output Arc from Transition SendInvitation to Place DeputyAgent_Out_Msg:

Expression: msgInvitationDeputy

- Output Arc from Transition SendInvitation to Place MonitorAgent_Out_Msg:

Expression: msgMakePreCommitment

- Transition SendInvitation:

Action:

```
(def invitation_content (vec (re-seq #"\w+" (. x get "content"))))

(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def invited_user_email (get (get quoted 0) 1))

(def msgInvitationDeputy (new nzdis.agent.message.Message "REQUEST" ( apply str "(crime_confirmation_invitation :invitation_id " (get invitation_content 2) ")")))

(.msgInvitation setSender (. agent_owner getAgentName))

(.msgInvitation setReceiver (. agent_owner getDeputyReference invited_user_email))

)

(def proposition (apply str (. agent_owner getDeputyReference invited_user_email) "_Confirmation_on_report(" (get invitation_content 2) ")_is_made_within_15_days"))
```

```
(. agent_owner createDeadline proposition (get
invitation_content 2) (. agent_owner getDeputyReference
invited_user_email))

(def pre_commitment_content (vec (re-seq #"\w+" (. x get
"content"))))

(def msgMakePreCommitment (new nzdis.agent.message.Message
"REQUEST" ( apply str "(make_commitment :creditor
'InstitutionAgent@OtagoPlatform' :debtor '" (. agent_owner
getDeputyReference invited_user_email) "' :condition '"(.
agent_owner getDeputyReference invited_user_email
"_Invitation(" (get invitation_content 2)
")_has_status_accepted' :content '" (. agent_owner
getDeputyReference invited_user_email)
"_Confirmation_on_report(" (get invitation_content 2)
")_is_made_within_15_days')")))

(.msgMakePreCommitment setSender (. agent_owner
getAgentName))

(.msgMakePreCommitment setReceiver
"MonitorAgent@OtagoPlatform")

)
```

1.11. Transition `Update_Modelchecker_by_DBAdmAgent`

- Message content example: `(update_modelchecker :truthvalue true :fact 'fact1')`

- Variable `result_update` example: `"Commitment01 active Commitment02 fulfilled Commitment03 violated"`

- Input Arc from Place `DBAdmAgent_Inc_Msg` to Transition `Update_Modelchecker_by_DBAdmAgent`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "Update_Modelchecker" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Update_Modelchecker_by_DBAdmAgent` to Place `MonitorAgent_Out_Msg`:

Expression: `msgMonitorUpdateByDBAdmAgent`

- Transition `Update_Modelchecker_by_DBAdmAgent`:

Action:

```
(def update_content (vec (re-seq #"\w+" (. x get
"content"))))

(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def incoming_fact (get (get quoted 0) 1))

(def fact (apply str ((. x getSender) "_" incoming_fact)))

(def result_update (. agent_owner updateModelInsertFact fact
(get update 2)))

(def  msgMonitorUpdateByDBAdmAgent (new
nzdis.agent.message.Message "INFORM" ( apply str
"(set_commitment :updates (" (apply str result_upate) ")")))

(.  msgMonitorUpdateByDBAdmAgent setSender (. agent_owner
getAgentName))

(.  msgMonitorUpdateByDBAdmAgent setReceiver
"MonitorAgent@OtagoPlatform")
```

1.12. Transition `Apply_Sanction_or_Reward`

- Message content example: `(apply_sanctionreward type: sanction :resource membership :deputy 'RegisteredUser01@OtagoPlatform')`

- Input Arc from Place `ReputationAgent_Inc_Msg` to Transition `Apply_Sanction_or_Reward`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "apply_sanctionreward" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Apply_Sanction_or_Reward` to Place `Expeling_Orders`:

Expression: `expelingOrder`

- Transition `Apply_Sanction_or_Reward`:

Action:

```
(def expelingOrder nil)

(def sanction_reward_content (vec (re-seq #"\w+" (. x get
"content"))))

(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def deputy (get (get quoted 0) 1))

(if (= "reward" (get sanction_reward_content 2))

(

(. agent_owner applyReward (get sanction_reward_content 4)
deputy)

)

(if (and (= "sanction" (get sanction_reward_content 2)) (=
"membership" (get sanction_reward_content 4)))

(

(def expelingOrder (apply str  "(expel :deputy " deputy ")"))

)

(

(. agent_owner applySanction deputy (get
sanction_reward_content 4))

))
```

1.13. Transition `Unregistration_Order`

- Token example: `"(expel :deputy 'RegisteredUser01@OtagoPlatform')"`

- Input Arc from Place `Expeling_Orders` to Transition `Unregistration_Order`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "expel" (first (re-seq #"\w+" x)) true false))`

Expression: `var("x");`

- Output Arc from Transition `Unregistration_Order` to Place `Ext_Agent_Out_Msg`:

Expression: `msgExternalAgentExpelingComunicate`

- Transition `Unregistration_Order`:

Action:

```
(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def deputy (get (get quoted 0) 1))

(. agent_owner expelAgent deputy)

(def msgExternalAgentExpelingComunicate (new
nzdis.agent.message.Message "INFORM" "you_were_expelled"))

(. msgExternalAgentExpelingComunicate setSender (.
agent_owner getAgentName))

(. msgExternalAgentExpelingComunicate setReceiver (.
agent_owner getExternalReference deputy))
```

2. Annotations of the Monitor Agent Workflow Coloured Petri Net (Figure 5.3)

2.1. Transition `Set_Environment`

- From Place `Initialization` to Transition `Set_Environment`:

Guard: `(instance? nzdis.agent.OpalAgent i)`

Expression: `var("i");`

- Transition `Set_Environment`:

Action: `(def agent_owner i)`

2.2. Transition `Make_Unset_Commitment_by_InstitutionAgent`

- Message content example: `(make_commitment :creditor 'InstitutionAgent@OtagoPlatform' :debitor 'RegisteredUser02@OtagoPlatform' :condition 'condition_formula' :content 'content_formula')`

- Input Arc from Place `InstitutionAgent_Inc_Msg` to Transition `Make_Unset_Commitment_by_InstitutionAgent`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "make_commitment" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Make_Unset_Commitment_by_InstitutionAgent` to Place `DeputyAgent_Out_Msg`:

Expression: `msgDeputyCommitmentProposal`

- Transition `Make_Unset_Commitment_by_InstitutionAgent`:

Action:

```
(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def creditor (get (get quoted 0) 1))

(def debitor (get (get quoted 1) 1))

(def condition_formula (get (get quoted 2) 1))

(def content_formula (get (get quoted 3) 1))

(def commitmentID (. agent_owner make_commitment "unset"
creditor debitor condition_formula content_formula))

(def  msgDeputyCommitmentProposal (new
nzdis.agent.message.Message "REQUEST" (apply str
"(commitment_proposal :commitmentID " commitmentID "
:commitment_creditor '" creditor "' :commitment_condition '"
condition_formula "' :commitment_content '" condition_formula
"')")))

(.  msgDeputyCommitmentProposal setSender (. agent_owner
getAgentName))

(.  msgDeputyCommitmentProposal setReceiver debitor)
```

2.3. Transition `Process_Unregistration`

- Message content example: `(unregistration_request :deputy 'RegisteredUser01@OtagoPlatform')`

- Input Arc from Place `InstitutionAgent_Inc_Msg` to Transition `Process_Unregistration`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "apply_sanction" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Process_Unregistration` to Place `InstitutionAgent_Out_Msg`:

Expression: `msgQueryModelChecker`

- Transition `Process_Unregistration`:

Action:

```
(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def deputy (get (get quoted 0) 1))

(def deputy_active_commitments (. agent_owner
processCommitments deputy)

(def  msgQueryModelChecker (new nzdis.agent.message.Message
"REQUEST" (apply str "(evaluate_current_state :commitments ("
deputy_active_commitments ")")))

(.  msgQueryModelChecker setSender (. agent_owner
getAgentName))

(.  msgQueryModelChecker setReceiver
"InstitutionAgent@OtagoPlatform")
```

2.4. Transition `Make_Compulsory_Commitment`

- Message content example: `(make_commitment_compulsory :creditor 'InstitutionAgent@OtagoPlatform' :debitor 'RegisteredUser02@OtagoPlatform' :condition 'condition_formula' :content 'content_formula')`

- Input Arc from Place `InstitutionAgent_Inc_Msg` to Transition `Make_Compulsory_Commitment`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= " make_commitment_compulsory" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Make_Compulsory_Commitment` to Place `InstitutionAgent_Out_Msg`:

Expression: `msgUpdateModelCheckerCompulsoryCommitment`

- Transition `Make_Compulsory_Commitment`:

Action:

```
(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def creditor (get (get quoted 0) 1))

(def debitor (get (get quoted 1) 1))

(def condition_formula (get (get quoted 2) 1))

(def content_formula (get (get quoted 3) 1))

(def commitmentID (. agent_owner make_commitment "pending"
creditor debitor condition_formula content_formula))

(def  msgUpdateModelCheckerCompulsoryCommitment (new
nzdis.agent.message.Message "REQUEST" (apply str
"(update_modelchecker_insert :commitment " commitmentID "
:condition_formula '" condition_formula "' :content_formula
'" content_formula "')")))

(.  msgUpdateModelCheckerCompulsoryCommitment setSender (.
agent_owner getAgentName))

(.  msgUpdateModelCheckerCompulsoryCommitment setReceiver
"InstitutionAgent@OtagoPlatform")
```

2.5. Transition `Process_ModelChecker_Evaluation`

- Message content example: `(query_responce :commitments (Commitment01 content_false Commitment02 content_true))`
- Message content example: `(query_responce_activate :commitments (Commitment01 condition_false Commitment02 condition_true))`

- Input Arc from Place `InstitutionAgent_Inc_Msg` to Transition `Process_ModelChecker_Evaluation`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (or ((= "query_responce" (first (re-seq #"\w+" (. x get`

```
"content"))) true false) (= "query_responce_activate" (first
(re-seq #"\w+" (. x get "content"))) true false))
```

Expression: `var("x");`

- Output Arc from Transition `Process_ModelChecker_Evaluation` to Place `SetCommitment`:

Expression: `commandSetCommitment`

- Transition `Process_ModelChecker_Evaluation`:

Action:

```
(def query_result (vec (re-seq #"\w+" (. x get "content"))))

(def setCommitment nil)

(def eval_result (rest (rest query_result)))

(def helper (repeat (count eval_result) " "))

(def evaluation_result (apply str (interleave eval_result
helper)))

(if (= "query_responce" (first (re-seq #"\w+" (. x get
"content"))) true false)

(def setCommitment (apply str
"(setCommitment_fulfilled_violated :evaluation_results ("
evaluation_result "))"))

)

(if (= "query_responce_activate" (first (re-seq #"\w+" (. x
get "content"))) true false)

(def setCommitment (apply str "(setCommitment_active
:evaluation_results (" evaluation_result "))"))

)
```

2.6. Transition `Set_Commitment_Cancelled_Timeout`

- Message content example: `(timeout_expired :commitments (Commitment01 Commitment02 Commitment03))`

- Input Arc from Place `InstitutionAgent_Inc_Msg` to Transition `Set_Commitment_Cancelled_Timeout`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "timeout_expired" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Transition `Set_Commitment_Cancelled_Timeout`:

Action:

```
(def timeout (vec (re-seq #"\w+" (. x get "content"))))

((def commitments_expired (rest (rest query_result)))

(for [in commitments_expired] (. setCommitmentExpired in))
```

2.7. Transition `Set_Commitment_Active`

- Command example: `(setCommitment_Active :evaluation_results (Commitment01 condition_true Commitment02 condition_false))`

- Input Arc from Place `Set_Commitment` to Transition `Set_Commitment_Active`:

Guard: `(and (instance? String x)(if (= "setCommitment_Active" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Transition `Set_Commitment_Active`:

Action:

```
(def query_result (vec (re-seq #"\w+" (. x get "content"))))

(def evaluation_result (rest (rest query_result)))

(def commitmentIDs "")

(def commitment_condition_evaluation "")

(defn function_separate [i] (if (= (first i) \C) (def commitmentIDs (apply str commitmentIDs i " ")) (def
```

```
commitment_condition_evaluation (apply str
commitment_condition_evaluation i " "))))

(for [in evaluation_result] (function_separate in))

(def commIDs (vec (re-seq #"\w+" commitmentIDs)))

(def condition_evaluation (vec (re-seq #"\w+"
commitment_condition_evaluation)))

(def quantity (range (count commIDs)))

(for (i quantity) ((. setCommitment (get commIDs i) (get
condition_evaluation i)))
```

2.8. Transition `Set_Commitment_Fulfilled_or_Violated`

- Command example: `(setCommitment_fulfilled_violated :evaluation_results (Commitment01 content_true Commitment02 content_false Commitment03 violated))`

- Input Arc from Place `Set_Commitment` to Transition `Set_Commitment_Fulfilled_or_Violated`:

Guard: `(and (instance? String x)(if (= "setCommitment_fulfilled_violated" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Set_Commitment_Fulfilled_or_Violated` to Place `ReputationAgent_Out_Msg`:

Expression: `messagesReputationUpdate`

- Transition `Set_Commitment_Fulfilled_or_Violated`:

Action:

```
(def query_result (vec (re-seq #"\w+" (. x get "content"))))

(def evaluation_result (rest (rest query_result)))

(def commitmentIDs "")

(def commitment_content_evaluation "")
```

```
(defn function_separate [i] (if (= (first i) \C) (def
commitmentIDs (apply str commitmentIDs i " ")) (def
commitment_content_evaluation (apply str
commitment_content_evaluation i " "))))

(for [in evaluation_result] (function_separate in))

(def commIDs (vec (re-seq #"\w+" commitmentIDs)))

(def content_evaluation (vec (re-seq #"\w+"
commitment_content_evaluation)))

(def quantity (range (count commIDs)))

(for (i quantity) ((. setCommitment (get commIDs i) (get
content_evaluation i)))

(def messagesReputationUpdate ())

(for (i quantity) (

(def  msgReputationUpdate (new nzdis.agent.message.Message
"REQUEST" (apply str "(update_reputation
:commitment_evaluation " (get content_evaluation i) " :debtor
" (. agent_owner getDebtor (get commIDs i)) ")" )))

(.  msgReputationUpdate setSender (. agent_owner
getAgentName))

(.  msgReputationUpdate setReceiver
"ReputationAgent@OtagoPlatform")

(def msg (list msgReputationUpdate))

(concat messagesReputationUpdate msg)

)
```

2.9. Transition `Query_Commitment_Repository`

- Message content example: `(query_commitment_repository :deputy 'RegisteredUser01@OtagoPlatform')`

- Input Arc from Place `InstitutionAgent_Inc_Msg` to Transition `Query_Commitment_Repository`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "query_commitment_repository" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Query_Commitment_Repository` to Place `InstitutionAgent_Out_Msg`:

Expression: `msgQueryResult`

- Transition `Query_Commitment_Repository`:

Action:

```
(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def deputy (get (get quoted 0) 1))

(def  msgQueryResult (new nzdis.agent.message.Message
"INFORM" (apply str "(query_responce (" (. agent_owner
queryCommitmentRepository deputy) ")")))

(.  msgQueryResult setSender (. agent_owner getAgentName))

(.  msgQueryResult setReceiver (. x getSender))
```

2.10. Transition `Make_Unset_Commitment`

- Message content example: `(make_commitment :creditor 'RegisteredUser01@OtagoPlatform' :debitor 'RegisteredUser02@OtagoPlatform' :condition 'condition_formula' :content 'content_formula')`

- Input Arc from Place `DeputyAgent_Inc_Msg` to Transition `Make_Unset_Commitment`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "make_commitment" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Make_Unset_Commitment` to Place `DeputyAgent_Out_Msg`:

Expression: `msgCommitmentProposal`

- Transition `Make_Unset_Commitment`:

Action:

```
(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def creditor (get (get quoted 0) 1))

(def debitor (get (get quoted 1) 1))

(def condition_formula (get (get quoted 2) 1))

(def content_formula (get (get quoted 3) 1))

(def commitmentID (. agent_owner make_commitment "unset"
creditor debitor condition_formula content_formula))

(def  msgCommitmentProposal (new nzdis.agent.message.Message
"REQUEST" (apply str "(commitment_proposal :commitmentID "
commitmentID " :commitment_creditor '" creditor "'
:commitment_condition '" condition_formula "'
:commitment_content '" content_formula "'")))

(.  msgCommitmentProposal setSender (. agent_owner
getAgentName))

(.  msgCommitmentProposal setReceiver debitor)
```

2.11. Transition `Set_Commitment_Cancelled_Directely`

- Message content example: `(cancel_commitment :commitmentID Commitment01)`

- Input Arc from Place `DeputyAgent_Inc_Msg` to Transition `Set_Commitment_Cancelled_Directely`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "apply_sanction" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Set_Commitment_Cancelled_Directely` to Place `InstitutionAgent_Out_Msg`:

Expression: `msgModelCheckerUpdateDeputyCancelCommitment`

- Transition `Set_Commitment_Cancelled_Directely`:

Action:

```
(def content (vec (re-seq #"\w+" (. x get "content"))))
```

```
(def commitment (. agent_owner getCommitment (get content
2)))

(. agent_owner setCommitmentCancel (get content 2))

(def msgModelCheckerUpdateDeputyCancelCommitment (new
nzdis.agent.message.Message "REQUEST" (apply str
"(update_modelchecker_cancel :commitment " (. commitment
getID) " :condition_formula '" (. commitment getCondition) "'
:content_formula '" (. commitment getContent) "')")))

(. msgModelCheckerUpdateDeputyCancelCommitment setSender (.
agent_owner getAgentName))

(. msgModelCheckerUpdateDeputyCancelCommitment setReceiver
"InstitutionAgent@OtagoPlatform")
```

2.12. Transition `Set_Commitment_Pending_or_Cancelled`

- Message content example: `(commitment_proposal_answer :commitmentID Commitment01 :outcome accept)`

- Input Arc from Place `DeputyAgent_Inc_Msg` to Transition `Set_Commitment_Pending_or_Cancelled`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "commitment_proposal_answer" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Set_Commitment_Pending_or_Cancelled` to Place `InstitutionAgent_Out_Msg`:

Expression: `msgModelCheckerUpdatePendingCancelled`

- Transition `Set_Commitment_Pending_or_Cancelled`:

Action:

```
(def content (vec (re-seq #"\w+" (. x get "content"))))

(if (= ("accept" (get content 4))

(

(. agent_owner setCommitmentPending (get content 2))
```

```
(def commitment (. agent_owner getCommitment (get content
2)))

(def  msgModelCheckerUpdatePendingCancelled (new
nzdis.agent.message.Message "REQUEST" (apply str
"(update_modelchecker_insert :commitment " (. commitment
getID) " :condition_formula '" (. commitment getCondition) "'
:content_formula '" (. commitment getContent) "')")))

(.  msgModelCheckerUpdatePendingCancelled setSender (.
agent_owner getAgentName))

(.  msgModelCheckerUpdatePendingCancelled setReceiver
"InstitutionAgent@OtagoPlatform")

)

(def msgModelCheckerUpdatePendingCancelled nil)

(. agent_owner setCommitmentCancel (get content 2))

)
```

2.13. Transition `Assign_Commitments`

- Message content example: `(assign_commitments :current_debtor_deputy 'RegisteredUser01@OtagoPlatform' :new_debitor_deputy 'RegisteredUser02@OtagoPlatform')`

- Input Arc from Place `InstitutionAgent_Inc_Msg` to Transition `Assign_Commitmentss`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "assign_commitments" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Assign_Commitments` to Place `InstitutionAgent_Out_Msg`:

Expression: `msgModelCheckerUpdateAssign`

- Transition `Assign_Commitments`:

Action:

```
(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))
```

```
(def old_debtor (get (get quoted 0) 1))

(def new_debtor (get (get quoted 1) 1))

(def commitments (. agent_owner assignCommitments old_debtor
new_debtor))

(def  msgModelCheckerUpdateAssign (new
nzdis.agent.message.Message "REQUEST" (apply str
"(update_modelchecker_assign :old_debtor '" old_debtor "'
:new_debtor '" new_debtor "' :commitmentIDs (" (apply str
commitments) ")")))

(.  msgModelCheckerUpdateAssign setSender (. agent_owner
getAgentName))

(.  msgModelCheckerUpdateAssign setReceiver
"InstitutionAgent@OtagoPlatform")
```

3. Annotations of the Reputation Agent Workflow Coloured Petri Net (Figure 5.4)

3.1. Transition `Set_Environment`

- From Place `Initialization` to Transition `Set_Environment`:

Guard: `(instance? nzdis.agent.OpalAgent i)`

Expression: `var("i");`

- Transition `Set_Environment`:

Action: `(def agent_owner i)`

3.2. Transition `Create_Record`

- Message content example: `(create_file :external_email 'teste.wikicrimes@gmail.com' :deputy 'DeputyRegisteredUser01@OtagoPlatform')`

- Input Arc from Place `InstitutionAgent_Inc_Msg` to Transition `Create_Record`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "create_file" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Transition `Create_Record`:

Action:

```
(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def external_email (get (get quoted 0) 1))

(def deputy (get (get quoted 1) 1))

(. agent_owner createRecord external_email deputy)
```

3.3. Transition `Reputation_Update`

- Message content example: `(update_reputation :type negative :agent 'RegisteredUserDeputy01@OtagoPlatform')`

- Input Arc from Place `MonitorAgent_Inc_Msg` to Transition `Reputation_Update`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "update_reputation" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Reputation_Update` to Place `InstitutionAgent_Out_Msg`:

Expression: `msgSanctionRewardMonitorAgent`

- Output Arc from Transition `Reputation_Update` to Place `DeputyAgent_Out_Msg`:

Expression: `msgSanctionReward`

- Transition `Reputation_Update`:

Action:

```
(def update_record (vec (re-seq #"\w+" (. x get "content"))))

(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def deputy (get (get quoted 0) 1))

(def result_update (. agent_owner updateRecord (get update_record 2) deputy))
```

```
(def msgSanctionRewardMonitorAgent (new
nzdis.agent.message.Message "INFORM" ( apply str
"(apply_sanctionreward :deputy '" deputy "' :type " (.
result_update getType) " :resource " (. result_update
getResource) ")")))

(. msgSanctionRewardMonitorAgent setSender (. agent_owner
getAgentName))

(. msgSanctionRewardMonitorAgent setReceiver
"InstitutionAgent@OtagoPlatform")

(def msgSanctionReward (new nzdis.agent.message.Message
"INFORM" ( apply str "(sanctionreward_inform type: " (.
result_update getType) " :resource " (. result_update
getResource) ")")))

(. msgSanctionReward setSender (. agent_owner getAgentName))

(. msgSanctionReward setReceiver deputy)
```

3.4. Transition `Reputation_Query_By_Deputy`

- Message content example: `(reputation_query_deputy :deputy 'RegisteredUser01@OtagoPlatform')`

- Input Arc from Place `DeputyAgent_Inc_Msg` to Transition `Reputation_Query_By_Deputy`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "reputation_query_deputy" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Reputation_Query_By_Deputy` to Place `Queries`:

Expression: `DeputyQuery`

- Transition `Reputation_Query_By_Deputy`:

Action:

```
(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def deputy (get (get quoted 0) 1))

(def query_result (. agent_owner queryRecord deputy))
```

```
(def DeputyQuery (apply str "(deputy_query :query_result "
(. query_result getReputation) " :deputy '" (. query_result
getDeputy) "')"))
```

3.5. Transition `Reputation_Query_By_DatabaseAdmAgent`

- Message content example: `(reputation_query_databaseadm :deputy 'RegisteredUser01@OtagoPlatform')`

- Input Arc from Place `DatabaseAdmAgent_Inc_Msg` to Transition `Reputation_Query_By_DatabaseAdmAgent`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "reputation_query_databaseadm" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Reputation_Query_By_DatabaseAdmAgent` to Place `Queries`:

Expression: `DatabaseAdmQuery`

- Transition `Reputation_Query_By_DatabaseAdmAgent`:

Action:

```
(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def deputy (get (get quoted 0) 1))

(def query_result (. agent_owner queryRecord deputy))

(def DatabaseAdmQuery (apply str "(DatabaseAdm_query
:query_result " (. query_result getReputation) " :deputy '"
(. query_result getDeputy) "' :source " (. x getSender)
"')"))
```

3.6. Transition `Query_Process`

- Command example: `(DatabaseAdm_query :query_result 'query_result' :deputy 'RegisteredUser01@OtagoPlatform' :source 'RegisteredUser05@OtagoPlatform')`

- Input Arc from Place `Queries` to Transition `Query_Process`:

Guard: `(and (instance? String x) (or (if (= "DatabaseAdm_query" (first (re-seq #"\w+" (. x get "content"))) true false) (if (= "deputy_query" (first (re-seq #"\w+" (. x get "content"))) true false)))`

Expression: `var("x");`

- Output Arc from Transition `Query_Process` to Place `DeputyAgent_Out_Msg`:

Expression: `msgQueryResultDeputy`

- Output Arc from Transition `Query_Process` to Place `DatabaseAdmAgent_Out_Msg`:

Expression: `msgQueryResultDatabaseAdmAgent`

- Transition `Query_Process`:

Action:

```
(def  msgQueryResultDeputy nil)

(def  msgQueryResultDatabaseAdmAgent nil)

(def query_command (vec (re-seq #"\w+" (. x get "content"))))

(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def reputation (get (get quoted 0) 1))

(def owner (get (get quoted 1) 1))

(def source (get (get quoted 2) 1))

(if (= "DatabaseAdm_query" (get query_comand 0)

(

(def  msgQueryResultDatabaseAdmAgent (new
nzdis.agent.message.Message "INFORM" ( apply str
"(reputation_query_result :reputation '" reputation "'
:deputy '" owner "')")))

(.  msgQueryResultDatabaseAdmAgent setSender (. agent_owner
getAgentName))
```

```
(. msgQueryResultDatabaseAdmAgent setReceiver
"DatabaseAdmAgent@OtagoPlatform")

)

(if (= "deputy_query" (get query_comand 0)

(

(def msgQueryResultDeputy (new nzdis.agent.message.Message
"INFORM" ( apply str "(reputation_query_result :reputation '"
reputation "' :deputy '" owner "')")))

 (. msgQueryResultDeputy setSender (. agent_owner
getAgentName))

(. msgQueryResultDeputy setReceiver source)

)
```

3.7. Transition `Reputation_Update_by_DatabaseAdmAgent`

- Message content example: `(update_reputation :type positive :agent 'RegisteredUserDeputy01@OtagoPlatform')`

- Input Arc from Place `DatabaseAdmAgent_Inc_Msg` to Transition `Reputation_Update_by_Database_Adm_Agent`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "update_reputation" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Reputation_Update_by_Database_Adm_Agent` to Place `InstitutionAgent_Out_Msg`:

Expression: `msgSanctionRewardDatabaseAdmAgent`

- Output Arc from Transition `Reputation_Update_by_Database_Adm_Agent` to Place `DeputyAgent_Out_Msg`:

Expression: `msgSanctionReward`

- Transition `Reputation_Update_by_Database_Adm_Agent`:

Action:

```
(def update_record (vec (re-seq #"\w+" (. x get "content"))))

(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def deputy (get (get quoted 0) 1))

(def result_update (. agent_owner updateRecord (get
update_record 2) deputy))

(def  msgSanctionRewardDatabaseAdmAgent (new
nzdis.agent.message.Message "INFORM" ( apply str
"(apply_sanctionreward :deputy '" deputy "type: "' (.
result_update getType) " :resource " (. result_update
getResource) ")")))

(.  msgSanctionRewardDatabaseAdmAgent setSender (.
agent_owner getAgentName))

(.  msgSanctionRewardDatabaseAdmAgent setReceiver
"InstitutionAgent@OtagoPlatform")

(def  msgSanctionReward (new nzdis.agent.message.Message
"INFORM" ( apply str "(sanctionreward_inform type: " (.
result_update getType) " :resource " (. result_update
getResource) ")")))

(.  msgSanctionReward setSender (. agent_owner getAgentName))

(.  msgSanctionReward setReceiver deputy)
```

4. Annotations of the Database Administrator Agent Workflow Coloured Petri Net (Figure 5.5)

4.1. Transition `Set_Environment`

- From Place `Initialization` to Transition `Set_Environment`:

Guard: `(instance? nzdis.agent.OpalAgent i)`

Expression: `var("i");`

- Transition `Set_Environment`:

Action: `(def agent_owner i)`

4.2. Transition `Comment_System`

- Message content example: `(comment_system :comment 'comment')`

- Input Arc from Place `DeputyAgent_Inc_Msg` to Transition `Comment_System`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "comment_system" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Transition `Coment_System`:

Action:

```
(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def comment (get (get quoted 0) 1))

 (. agent_owner makeComment comment)
```

4.3. Transition `Query_Databese`

- Message content example: `(query_databese :query 'query')`

- Input Arc from Place `DeputyAgent_Inc_Msg` to Transition `Query_Database`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "query_database" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Query_Database` to Place `DeputyAgent_Out_Msg`:

Expression: `msgQueryDatabaseResult`

- Transition `Query_Database`:

Action:

```
(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))
```

```
(def query (get (get quoted 0) 1))

(def query_result (. agent_owner makeQuery query))

(def  msgQueryDatabaseResult (new nzdis.agent.message.Message
"INFORM" ( apply str "(database_query_result :result "
query_result ")")))

(.  msgQueryDatabaseResult setSender (. agent_owner
getAgentName))

(.  msgQueryDatabaseResult setReceiver (. x getSender))
```

4.4. Transition `Comment_CriminalFact`

- Message content example: `(comment_criminal_fact :criminal_factID 2236 :comment 'comment')`

- Input Arc from Place `InstitutionAgent_Inc_Msg` to Transition `Create_Record`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "comment_criminal_fact" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Transition `Comment_CriminalFact`:

Action:

```
(def content (vec (re-seq #"\w+" (. x get "content"))))

(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def comment (get (get quoted 0) 1))

 (. agent_owner makeCommentCriminalFact (get content 2)
comment)
```

4.5. Transition `Register_CriminalFact`

- Message content example: `(register_crime :crime_type theft :latitude '999.999' :longitude '888.888' :confirming_agent 'test.wikicrimes@gmail.com')`

- Input Arc from Place `DeputyAgent_Inc_Msg` to Transition `Register_CriminalFact`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "register_crime" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Register_CriminalFact` to Place `DeputyAgent_Out_Msg`:

Expression: `msgCrimeRegistrationResultDeputy`

- Output Arc from Transition `Register_CriminalFact` to Place `InstitutionAgent_Out_Msg`:

Expression: `messagesCrimeRegistrationResultInstitution`

- Transition `Register_CriminalFact`:

Action:

```
(def content (vec (re-seq #"\w+" (. x get "content"))))

(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def latitude (get (get quoted 0) 1))

(def longitude (get (get quoted 1) 1))

(def confirming_agent (get (get quoted 2) 1))

(def registration_result (. agent_owner registerCrime (get content 2) latitude longitude confirming_agent))

(def  msgCrimeRegistrationResultDeputy (new nzdis.agent.message.Message "INFORM" ( apply str "(crime_registration_result :result " registration_result ")")))

(.  msgCrimeRegistrationResultDeputy setSender (. agent_owner getAgentName))

(.  msgCrimeRegistrationResultDeputy setReceiver (. x getSender))

(def messagesCrimeRegistrationResultInstitution ())
```

```
( if (not (= "0" crimeID)

(

(def alert_areas (. agent_owner verifyAreas crimeID))

(for (i alert_areas) (

(def msgCrimeRegistrationResultInstitution (new
nzdis.agent.message.Message "INFORM" ( apply str
"(observation_area_alert :observation_area_id" (. i
getObservationAreaID) " :crime_id " (. i getCrimeID) "
:agent_owner_email " (. i getOwnerEmail)")")))

(. msgCrimeRegistrationResultInstitution setSender (.
agent_owner getAgentName))

(. msgCrimeRegistrationResultInstitution setReceiver
"InstitutionAgent@OtagoPlatform")

(def msg (list msgCrimeRegistrationResultInstitution))

(concat messagesCrimeRegistrationResultInstitution msg)

))
```

4.6. Transition `Confirm_CriminalFact`

- Message content example: `(crime_confirmation :crime_id 1 :type negative :crime_reporter 'DeputyRegisteredUser01@OtagoPlatform')`

- Input Arc from Place `InstitutionAgent_Inc_Msg` to Transition `Confirm_CriminalFact`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "crime_confirmation" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Confirm_CriminalFact` to Place `InstitutionAgent_Out_Msg`:

Expression: `msgUpdateModelChecker`

- Output Arc from Transition `Confirm_CriminalFact` to Place `ReputationAgent_Out_Msg`:

Expression: `msgUpdateReputation`

- Transition `Confirm_CriminalFact`:

Action:

```
(def content (vec (re-seq #"\w+" (. x get "content"))))

(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def crime_reporter (get (get quoted 0) 1))

(. agent_owner confirmCrime (get content 2) (get content 4))

(def msgUpdateModelChecker nil)

(if (= "negative" (get content 4))

(

(def fact (apply str crime_reporter
"_Negative_confirmation_on_report(" (get content 2)
")_is_made")

(def  msgUpdateModelChecker (new nzdis.agent.message.Message
"REQUEST" ( apply str "(update_modelchecker :truthvalue true
:fact '" fact "')")))

(.  msgUpdateModelChecker setSender (. agent_owner
getAgentName))

(.  msgUpdateModelChecker setReceiver
"InstitutionAgent@OtagoPlatform")

))

(if (= "positive" (get content 4))

(

(def  msgUpdateReputation (new nzdis.agent.message.Message
"REQUEST" ( apply str "(update_reputation :type positive
:agent '" crime_reporter "')")))

(.  msgUpdateReputation setSender (. agent_owner
getAgentName))
```

```
(. msgUpdateReputation setReceiver
"ReputationAgent@OtagoPlatform")

)))
```

4.7. Transition Denounce_Abuse

- Message content example: `(abuse_denounce :crime_id 1 :crime_reporter 'DeputyRegisteredUser01@OtagoPlatform')`

- Input Arc from Place `InstitutionAgent_Inc_Msg` to Transition `Denounce_Abuse`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "abuse_denounce" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Denounce_Abuse` to Place `InstitutionAgent_Out_Msg`:

Expression: `msgUpdateModelCheckerAbuseDenounce`

- Transition `Denounce_Abuse`:

Action:

```
(def content (vec (re-seq #"\w+" (. x get "content"))))

(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def crime_reporter (get (get quoted 0) 1))

(. agent_owner denounceCrime (get content 2))

(def msgUpdateModelCheckerAbuseDenounce nil)

(def fact (apply str crime_reporter
"_Abuse_denounce_on_report(" (get content 2) ")_is_made")

(def msgUpdateModelCheckerAbuseDenounce (new
nzdis.agent.message.Message "REQUEST" ( apply str
"(update_modelchecker :truthvalue true :fact '" fact "')")))

(. msgUpdateModelCheckerAbuseDenounce setSender (.
agent_owner getAgentName))

(. msgUpdateModelCheckerAbuseDenounce setReceiver
"InstitutionAgent@OtagoPlatform")
```

5. Annotations of the Deputy Registered User Agent Workflow, Deputy Certifier Entity Workflow, Deputy Invited User Workflow and Deputy Browser User Workflow Coloured Petri Nets (Figures 5.7, 5.8, 5.10 and 5.11)

5.1. Transition `Set_Environment`

- From Place `Initialization` to Transition `Set_Environment`:

Guard: `(instance? nzdis.agent.OpalAgent i)`

Expression: `var("i");`

- Transition `Set_Environment`:

Action: `(def agent_owner i)`

5.2. Transition `Comment_Criminal_Fact`

- Message content example: `(comment_criminal_fact :criminal_factID 5525 :comment 'comment')`

- Input Arc from Place `Ext_Agent_Inc_Msg` to Transition `Comment_Criminal_Fact`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "comment_criminal_fact" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Comment_Criminal_Fact` to Place `Database_Adm_Agent_Out_Msg`:

Expression: `msgCommentCriminalFact`

- Transition `Comment_Criminal_Fact`:

Action:

```
(def content (. x get "content"))))

(def msgCommentCriminalFact (new nzdis.agent.message.Message
"REQUEST" content)))

(. msgCommentCriminalFact setSender (. agent_owner
getAgentName))
```

```
(. msgCommentCriminalFact setReceiver
"DatabaseAdmAgent@OtagoPlatform")
```

5.3. Transition `Register_Criminal_Fact`

- Message content example: `(register_crime :crime_type robbery :latitude '555.789' :longitude '234.123' :confirming_agent 'test.wikicrimes@gmail.com')`

- Input Arc from Place `Ext_Agent_Inc_Msg` to Transition `Register_Criminal_Fact`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "register_crime" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Register_Criminal_Fact` to Place `Database_Adm_Agent_Out_Msg`:

Expression: `msgRegisterCriminalFact`

- Transition `Register_Criminal_Fact`:

Action:

```
(def content (. x get "content"))))

(def msgRegisterCriminalFact (new
nzdis.agent.message.Message "REQUEST" content)))

(. msgRegisterCriminalFact setSender (. agent_owner
getAgentName))

(. msgRegisterCriminalFact setReceiver
"DatabaseAdmAgent@OtagoPlatform")
```

5.4. Transition `Denounce_Abuse`

- Message content example: `(abuse_denounce :crime_id 2 :crime_reporter 'DeputyRegisteredUser01@OtagoPlatform')`

- Input Arc from Place `Ext_Agent_Inc_Msg` to Transition `Denounce_Abuse`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "abuse_denounce" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Denounce_Abuse` to Place `Database_Adm_Agent_Out_Msg`:

Expression: `msgAbuseDenounce`

- Transition `Denounce_Abuse`:

Action:

```
(def content (. x get "content"))))

(def  msgAbuseDenounce (new nzdis.agent.message.Message "REQUEST" content)))

(.  msgAbuseDenounce setSender (. agent_owner getAgentName))

(.  msgAbuseDenounce setReceiver "DatabaseAdmAgent@OtagoPlatform")
```

5.5. Transition `Confirm_Criminal_Fact`

- Message content example: `(crime_confirmation :crime_id 2 :type positive :crime_reporter 'DeputyRegisteredUser01@OtagoPlatform')`

- Input Arc from Place `Ext_Agent_Inc_Msg` to Transition `Confirm_Criminal_Fact`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "crime_confirmation" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Confirm_Criminal_Fact` to Place `Database_Adm_Agent_Out_Msg`:

Expression: `msgConfirmCriminalFact`

- Transition `Confirm_Criminal_Fact`:

Action:

```
(def content (. x get "content"))))
```

```
(def msgConfirmCriminalFact (new nzdis.agent.message.Message
"REQUEST" content)))
```

```
(. msgConfirmCriminalFact setSender (. agent_owner
getAgentName))
```

```
(. msgConfirmCriminalFact setReceiver
"DatabaseAdmAgent@OtagoPlatform")
```

5.6. Transition Comment_System

- Message content example: `(comment_system :comment 'comment')`

- Input Arc from Place `Ext_Agent_Inc_Msg` to Transition `Comment_System`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "comment_system" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Comment_System` to Place `Database_Adm_Agent_Out_Msg`:

Expression: `msgCommentSystem`

- Transition `Comment_System`:

Action:

```
(def content (. x get "content"))))
```

```
(def msgCommentSystem (new nzdis.agent.message.Message
"REQUEST" content)))
```

```
(. msgCommentSystem setSender (. agent_owner getAgentName))
```

```
(. msgCommentSystem setReceiver
"DatabaseAdmAgent@OtagoPlatform")
```

5.7. Transition Query_Database

- Message content example: `(query_databese :query 'query')`

- Input Arc from Place `Ext_Agent_Inc_Msg` to Transition `Query_Database`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "query_database" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Query_Database` to Place `Database_Adm_Agent_Out_Msg`:

Expression: `msgQueryDatabase`

- Transition `Query_Database`:

Action:

```
(def content (. x get "content"))))

(def  msgQueryDatabase (new nzdis.agent.message.Message "REQUEST" content)))

(.  msgQueryDatabase setSender (. agent_owner getAgentName))

(.  msgQueryDatabase setReceiver "DatabaseAdmAgent@OtagoPlatform")
```

5.8. Transition `Reputation_Query`

- Message content example: `(reputation_query_deputy :deputy 'RegisteredUser01@OtagoPlatform')`

- Input Arc from Place `Ext_Agent_Inc_Msg` to Transition `Reputation_Query`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "reputation_query_deputy" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Reputation_Query` to Place `Reputation_Agent_Out_Msg`:

Expression: `msgReputationQueryDeputy`

- Transition `Reputation_Query`

Action:

```
(def content (. x get "content"))))
```

```
(def msgQueryDatabase (new nzdis.agent.message.Message
"REQUEST" content)))

(. msgQueryDatabase setSender (. agent_owner getAgentName))

(. msgQueryDatabase setReceiver
"ReputationAgent@OtagoPlatform")
```

5.9. Transition Accept_Refuse_Commitment

- Message content example: `(commitment_proposal_answer :commitmentID Commitment01 :outcome accept)`

- Input Arc from Place `Ext_Agent_Inc_Msg` to Transition `Accept_Refuse_Commitment`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "commitment_proposal_answer" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Accept_Refuse_Commitment` to Place `Monitor_Agent_Out_Msg`:

Expression: `msgAcceptRefuseCommitment`

- Transition `Accept_Refuse_Commitment`:

Action:

```
(def content (. x get "content"))))

(def msgQueryDatabase (new nzdis.agent.message.Message
"REQUEST" content)))

(. msgQueryDatabase setSender (. agent_owner getAgentName))

(. msgQueryDatabase setReceiver
"MonitorAgent@OtagoPlatform")
```

5.10. Transition Monitored_Interaction

- Message content example: `(make_commitment :debitor 'RegisteredUser02@OtagoPlatform' :condition 'condition_formula' :content 'content_formula')`

- Input Arc from Place `Ext_Agent_Inc_Msg` to Transition `Monitored_Interaction`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "make_commitment" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Monitored_Interaction` to Place `Monitor_Agent_Out_Msg`:

Expression: `msgMonitoredInteraction`

- Transition `Monitored_Interaction`:

Action:

```
(def content (vec (re-seq #"\w+" x)))

(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def debitor (get (get quoted 0) 1))

(def condition_formula (get (get quoted 1) 1))

(def content_formula (get (get quoted 2) 1))

(def  msgMonitoredInteraction (new
nzdis.agent.message.Message "REQUEST" (apply str
"(make_commitment :creditor " (. agent_owner
getAgentNameString) " :debitor " debitor " :condition '"
condition_formula "' :content '" content_formula "')")))

(.  msgMonitoredInteraction setSender (. agent_owner
getAgentName))

(.  msgMonitoredInteraction setReceiver
"MonitorAgent@OtagoPlatform")
```

5.11. Transition `Change_Role`

- Message content example: `(change_role :new_role CertifierEntity)`

- Input Arc from Place `Ext_Agent_Inc_Msg` to Transition `Change_Role`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "change_role" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Change_Role` to Place `Institution_Agent_Out_Msg`:

Expression: `msgChangeRole`

- Transition `Change_Role`:

Action:

```
(def content (vec (re-seq #"\w+" (. x get "content"))))

(def  msgChangeRole (new nzdis.agent.message.Message "REQUEST" (apply str "(change role :new_role " (get content 2) ":external_email '" (. agent_owner getEmail) "')")))

(.  msgChangeRole setSender (. agent_owner getAgentName))

(.  msgChangeRole setReceiver "InstitutionAgent@OtagoPlatform")
```

5.12. Transition `Unregistration`

- Message content example: `(unregistration_deputy)`

- Input Arc from Place `Ext_Agent_Inc_Msg` to Transition `Unregistration`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "unregistration_deputy" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Unregistration` to Place `Institution_Agent_Out_Msg`:

Expression: `msgUnregistration`

- Transition `Unregistration`:

Action:

```
(def content (. x get "content"))))
```

```
(def msgUnregistration (new nzdis.agent.message.Message
"REQUEST" content)))

(. msgUnregistration setSender (. agent_owner getAgentName))

(. msgUnregistration setReceiver
"InstitutionAgent@OtagoPlatform")
```

5.13. Transition Unmonitored_Interaction_Inc

- Message content example: `(free_speech :receiver 'RegisteredUser02@OtagoPlatform' :content 'Hello!')`

- Input Arc from Place `Ext_Agent_Inc_Msg` to Transition `Unmonitored_Interaction_Inc`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "free_speech" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Unmonitored_Interaction_Inc` to Place `DeputyAgent_Out_Msg`:

Expression: `msgReplyFreeSpeech`

- Transition `Unmonitored_Interaction_Foward`:

Action:

```
(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def receiver (get (get quoted 0) 1))

(def msgFreeSpeechContent (get (get quoted 1) 1))

(def msgFowardFreeSpeech (new nzdis.agent.message.Message
"REQUEST" (apply str "(free_speech :sender '" (. agent_owner
getAgentNameString "' :content '" msgFreeSpeechContent
"')")))

(. msgFowardFreeSpeech setSender (. agent_owner
getAgentName))

(. msgFowardFreeSpeech setReceiver receiver)
```

5.14. Transition Unmonitored_Interaction_Reply

- Message content example: `(free_speech :sender 'RegisteredUser02@OtagoPlatform' :content 'Hello!')`

- Input Arc from Place `DeputyAgent_Inc_Msg` to Transition `Unmonitored_Interaction_Out`

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "free_speech" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Unmonitored_Interaction_Out` to Place `Ext_Agent_Out_Msg`:

Expression: `msgReplyFreeSpeech`

- Transition `Unmonitored_Interaction_Out`:

Action:

```
(def quoted (vec (re-sec #"'(.*?)'" (. x get "content"))))

(def sender (get (get quoted 0) 1))

(def msgFreeSpeechContent (get (get quoted 1) 1))

(def  msgReplyFreeSpeech (new nzdis.agent.message.Message "REQUEST" (apply str "(free_speech :receiver '" sender "' :content '" msgFreeSpeechContent "')")))

(.  msgReplyFreeSpeech setSender (. agent_owner getAgentName))

(.  msgReplyFreeSpeech setReceiver (. agent_owner getExternalReference))
```

5.15. Transition Analyse_Invitation

- Message content example: `(crime_confirmation_invitation :invitation_id 1)`

- Input Arc from Place `Institution_Agent_Inc_Msg` to Transition `Analyse_Invitation`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "crime_confirmation_invitation" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Analyse_Invitation` to Place `DeputyAgent_Out_Msg`:

Expression: `msgQueryDatabaseResult`

- Output Arc from Transition `Analyse_Invitation` to Place `Monitor_Agent_Out_Msg`:

Expression: `msgCrimeConfirmationInvitation`

- Transition `Analyse_Invitation`:

Action:

```
(def content (. x get "content"))

(def msgCrimeConfirmationInvitation (new
nzdis.agent.message.Message "REQUEST" content ))

(. msgCrimeConfirmationInvitation setSender (. x getSender))

(. msgCrimeConfirmationInvitation setReceiver (. agent_owner
getExternalReference))
```

5.16. Transition `ProcessQuery`

- Message content example: `(database_query_result :result 'query_result')`

- Input Arc from Place `Database_Adm_Agent_Inc_Msg` to Transition `ProcessQuery`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "database_query_result" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `ProcessQuery` to Place `Ext_Agent_Out_Msg`:

Expression: `msgDatabaseQueryResult`

- Transition `ProcessQuery`:

Action:

```
(def content (. x get "content"))

(def msgDatabaseQueryResult (new nzdis.agent.message.Message
"REQUEST" content ))

(. msgDatabaseQueryResult setSender (. x getSender))

(. msgDatabaseQueryResult setReceiver (. agent_owner
getExternalReference))
```

5.17. Transition `Request_Accept_Refuse_Commitment`

- Message content example: `(commitment_proposal :commitmentID Commitment01 :commitment_creditor 'RegisteredUser02@OtagoPlatform' :commitment_condition 'conditin_formula' :commitment_content 'content_formula')`

- Input Arc from Place `MonitorAgent_Inc_Msg` to Transition `Request_Accept_Refuse_Commitment`:

Guard: `(and (instance? nzdis.agent.message.Message x)(if (= "commitment_proposal" (first (re-seq #"\w+" (. x get "content"))) true false))`

Expression: `var("x");`

- Output Arc from Transition `Request_Accept_Refuse_Commitment` to Place `Ext_Agent_Out_Msg`:

Expression: `msgCommitmentProposal`

- Transition `Request_Accept_Refuse_Commitment`:

Action:

```
(def content (. x get "content"))

(def msgCommitmentProposal (new nzdis.agent.message.Message
"REQUEST" content ))

(. msgCommitmentProposal setSender (. x getSender))

(. msgCommitmentProposal setReceiver (. agent_owner
getExternalReference))
```

5.18. Transition Reputation_Query_Responce

- Message content example: (reputation_query_result :reputation '3' :deputy 'RegisteredUser03@OtagoPlatform')

- Input Arc from Place ReputationAgent_Inc_Msg to Transition Reputation_Query_Responce:

Guard: (and (instance? nzdis.agent.message.Message x)(if (= "deputy_query" (first (re-seq #"\w+" (. x get "content"))) true false))

Expression: var("x");

- Output Arc from Transition Reputation_Query_Responce to Place Ext_Agent_Out_Msg:

Expression: msgReputationQueryResult

- Transition Reputation_Query_Responce:

Action:

```
(def content (. x get "content"))

(def msgReputationQueryResult (new
nzdis.agent.message.Message "INFORM" content ))

(. msgReputationQueryResult setSender (. x getSender))

(. msgReputationQueryResult setReceiver (. agent_owner
getExternalReference))
```

5.19. Transition Deal_with_Sanction_Reward

- Message content example: (sanctionreward_inform :type sanction :resource resource)

- Input Arc from Place ReputationAgent_Inc_Msg to Transition Deal_with_Sanction_Reward:

Guard: (and (instance? nzdis.agent.message.Message x)(if (= "sanctionreward_inform" (first (re-seq #"\w+" (. x get "content"))) true false))

Expression: var("x");

- Output Arc from Transition `Deal_with_Sanction_Reward` to Place `Ext_Agent_Out_Msg`:

Expression: `msgSanctionRewardInform`

- Transition `Deal_with_Sanction_Reward`:

Action:

```
(def content (. x get "content"))

(def  msgSanctionRewardInform (new
nzdis.agent.message.Message "INFORM" content ))

(.  msgSanctionRewardInform setSender (. x getSender))

(.  msgSanctionRewardInform setReceiver (. agent_owner
getExternalReference))
```