Alma Mater Studiorum · Università di Bologna

Dottorato di Ricerca
in
Ingegneria Elettronica, Informatica e delle Telecomunicazioni

Cycle XXII
Disciplinary Sector:
ING-INF/05

Designing and Programming Organizational Infrastructures for Agents situated in Artifact-based Environments

Candidate:
Dott. Ing. Michele Piunti

Coordinator:
Chiar.mo Prof. Ing. Paola Mello

Supervisor:
Chiar.mo Prof. Ing. Antonio Natali

Co-Supervisors:
Ill.mo Prof. Ing. Alessandro Ricci
Ill.mo Prof. Ing. Andrea Omicini

Academic Year 2008 · 2009

to Agostino and Paola
Who bred wolves

and to Semola
She comes in colors everywhere

I call it the law of the instrument, and it may be formulated as follows: Give a small boy a hammer, and he will find that everything he encounters needs pounding.

ABRAHAM KAPLAN.
THE CONDUCT OF INQUIRY: METHODOLOGY FOR BEHAVIORAL SCIENCE, 1964.

We live in exactly one world, not two or three or seventeen.

JOHN R. SEARLE.
THE CONSTRUCTION OF SOCIAL REALITY, 1997.

I punti da A ad A sono costanti nati dal contrasto e per questo convergenti.

MARCO CASTOLDI.
DA A AD A, 2007.

Abstract

Actual trends in software development are pushing the need to face a multiplicity of diverse activities and interaction styles characterizing complex and distributed application domains, in such a way that the resulting dynamics exhibits some grade of order, i.e. in terms of evolution of the system and desired equilibrium. Autonomous agents and Multiagent Systems are argued in literature as one of the most immediate approaches for describing such a kind of challenges. Actually, agent research seems to converge towards the definition of renewed abstraction tools aimed at better capturing the new demands of open systems. Besides agents, which are assumed as autonomous entities purposing a series of design objectives, Multiagent Systems account new notions as first-class entities, aimed, above all, at modeling institutional/organizational entities, placed for normative regulation, interaction and teamwork management, as well as environmental entities, placed as resources to further support and regulate agent work.

The starting point of this thesis is recognizing that both organizations and environments can be rooted in a unifying perspective. Whereas recent research in agent systems seems to account a set of diverse approaches to specifically face with at least one aspect within the above mentioned, this work aims at proposing a unifying approach where both agents and their organizations can be straightforwardly situated in properly designed working environments. In this line, this work pursues reconciliation of environments with sociality, social interaction with environment based interaction, environmental resources with organizational functionalities with the aim to smoothly integrate the various aspects of complex and situated organizations in a coherent programming approach. Rooted in Agents and Artifacts (A&A) meta-model, which has been recently introduced both in the context of agent oriented software engineering and programming, the thesis promotes the notion of Embodied Organizations, characterized by computational infrastructures attaining a seamless integration between agents, organizations and environmental entities.

Preface

Engaging a PhD is a privilege to run across ideas, challenges, places, books, manuscripts, science, beautiful minds. There are many people who deserve special thanks for the support given along these years. I thank, first and foremost, my supervisor Alessandro Ricci, because without him nothing of this work would have been probably conceived. I thank him for his friendship, above all, and then for his continuous tension towards challenges, science, research, and for having shared so many ideas, drafts, white boards, pieces of paper, chinas, margaritas on the rocks, coffee machines... I will miss our *dadaist* brainstormings, Ale. Then I thank my supervisors, Antonio Natali and Andrea Omicini, for having instilled in me (and in us all) the gospel of *abstraction*, for having set the basis of what we are doing, and created an excellent *school* in computer science in the middle of Romagna. I thank all the people at ApiCe-lab in Cesena, Mirko Viroli, Andrea Santi, Matteo Casadei, Nazzareno Pompei, Marco Fabbri and finally a couple of wonderful creatures, Sara Montagna and Sofia Ricci, who you can often find together in the lab, after the nursery.

I am very thankful to Rino Falcone and Cristiano Castelfranchi at *Istituto di Scienze e Tecnologie della Cognizione - C.n.r.*, for having showed me the ropes of cognitive modeling, depicting so many ideas during my work at Institute of Cognitive Science and Technology. I thank them for having pushed further the notion of agency in computer science—years and years before this would became a recognized field. I thank all the other guys in Rome, Fabio Paglieri, Giovanni Pezzulo, Luca Tummolini, Tarek El Sheity, Raffaella Pocobello, Maria Miceli, Dimitri Ognibene and many others on the list, who always endure and resist, and have taught me what research is.

My thanks go to the group at *Ecoles des Mines* in St.Etienne, where I met awesome guys who have given me hospitality and showed me a new hope in the Massif Central, Loire. An immense thank goes to Olivier Boissier and Jomi Hübner, first of all, because they pushed so far a fruitful collaboration; I thank

them for having shared with me their ideas and models, and for having spent so much time in teaching me why I would need organizations, meeting again and again in a pivotal phase of this work. I thank Rosine Kitio, for her fundamental support and for her very important work, I thank France and I thank all the others researcher at G2I: Laurent Vercouter, Gauthier Picard, Yann Krupa, and many others who made me to feel at home in *Cours Fauriel, 42100*.

Finally, I want to thank all the people who has worked with me during the last years, to whom I owe inspiration and from whom I have learned so much. I thank my friend Emiliano Lorini and his *emotional* ideas, for having formalised so many exciting topics. I thank the *Jadex* group at Hamburg University, Lars Braubach, Alexander Pokahr and Winfried Lamersdorf, for their beautiful systems, and for their ceaseless support. I thank Antônio Carlos da Rocha Costa and Rafael Heitor Bordini, for having inspired important ideas at the basis of this work. I am very thankful to Juan Antonio Aguilar and Mehdi Dastani for their helpful comments and suggestions, and for having spent their time in reviewing a raw draft of this manuscript.

Heartfelt thanks go to my family, Agostino, Paola, Marco, to my grande dames, Teresa and Irma, and to my granddads, Guido and Salvatore. To Lou, who has driven me intothewild, and to Silvia, who deserves all that is going to happen henceforth.

Bologna, March 2010.

Contents

1 Introduction **1**
1.1 Global view and Objectives . . . 3
1.1.1 An Integrated approach to Agents, Environment, Organizations . . . 4
1.1.2 Objectives of this Thesis . . . 5
1.2 Overview of the Thesis . . . 7
1.2.1 Part I · Setting the Stage . . . 7
1.2.2 Part II · Developing Environment Infrastructures based on Artifacts . . . 8
1.2.3 Part III · Developing Organizational Infrastructures based on Artifacts . . . 9
1.2.4 Part IV · Agents, Organizations, Environment: a Unifying Approach . . . 10
1.3 Relevant Issues we do not address . . . 11

I Setting the Stage **13**

2 Organizations in MAS: Theories, Scopes and Directions **15**
2.1 Organizations in MAS . . . 15
2.2 Organization Oriented Programming . . . 17
2.2.1 Agent, Groups, Roles . . . 18
2.2.2 Collective Intentions . . . 20
2.2.3 Social Laws . . . 20
2.2.4 Electronic Institutions . . . 21
2.2.5 Normative Systems . . . 22
2.3 $\mathcal{M}$oise: an organizational model based on structural, functional and deontic dimensions . . . 26

2.3.1 Organizational Entities 28
2.3.2 Recasting Organizational Entities as Normative Systems . 30
2.3.3 Managing Organizations with a Normative Programming Language 31
2.4 Final Remarks on Organizations in MAS 35

3 Organizations Situated in MAS Environments 37
3.1 Situating Organizations in Computational Environments 37
3.2 Environments and Organizations in MAS 39
3.2.1 Current Approaches 39
3.2.2 Open Issues and Challenges 42
3.3 Environment as first class Abstraction in MAS 44
3.4 A Structured approach to Environments 47
3.4.1 Action Model 47
3.4.2 Perception Model 48
3.4.3 Computational Model 49
3.4.4 Internal Dynamics 51
3.4.5 Data Model (and Openness) 52
3.4.6 Distribution Model (and Localities) 52
3.5 Agents & Artifacts 53
3.5.1 Foundations 53
3.5.2 Meta-Model for engineering MAS 54
3.6 Final Remarks on Situated Organizations 59

II Developing Environment Infrastructures based on Artifacts 61

4 Environment Programming in CArtAgO 63
4.1 Taking the Environment Programming Perspective 63
4.2 Artifact-Based Environments 65
4.2.1 Artifact Computational Model 66
4.2.2 Actions to Work with Artifacts 68
4.2.3 Actions to Enter and Leave Workspaces 72
4.3 Environment Programming in CArtAgO 73
4.3.1 Artifact Programming Model 73
4.3.2 Integration with Agent Programming Platforms 78
4.4 Agents at work in CArtAgO Environments 79

4.4.1 Agent Programming in *Jason* 79
4.4.2 Using simple Artifacts 80
4.4.3 Using Artifacts to Externalize Activities 82
4.5 Cognitive Use 82
4.5.1 Mapping Goals and Beliefs on Artifact Functions 83
4.5.2 Externalisation and Internalisation 88
4.6 Final Remarks on programming Agents and Artifacts 91

5 Artifact Based Environments: a Formal Model of CArtAgO 95
5.1 Formalising Artifact-Based Environments 95
5.2 Structures 96
5.2.1 Agent Configuration 96
5.2.2 Artifact Configuration 97
5.2.3 Workspace Configuration 102
5.2.4 Workspace Initial Configuration 105
5.2.5 MAS Configuration 106
5.3 Dynamics 106
5.3.1 Agent Execution Cycle 107
5.3.2 Artifacts Dynamics 107
5.3.3 Agent Perceptive Activities 114
5.3.4 Agents Joining and Leaving Workspaces 117
5.3.5 Environment Management and Inspection 119
5.3.6 Workspace Time Evolution 124
5.4 Final Remarks on the Formalisation 124

6 Extending CArtAgO with Intra-Workspace Dynamics 127
6.1 Specifying global dynamics inside workspaces 127
6.2 Shaping the problem 128
6.2.1 Programming approaches 129
6.2.2 Workspace Rules 131
6.3 Sintax 134
6.4 Dynamics 136
6.5 Workspace Programming Examples 142
6.5.1 A Counter Infrastructure 143
6.5.2 Producers Consumers 144
6.6 Final Remarks on Intra-Workspace Dynamics 146

III Developing Organizational Infrastructures based on Artifacts 149

7 Programming Organizations in Practice 151
7.1 Taking the Organization Programming Perspective 151
7.2 Using Moise for modeling a concrete Organizational Entity 152
7.2.1 Structural Specification 153
7.2.2 Functional Specification 156
7.2.3 Deontic Specification 160
7.3 From the Moise specification to a Normative Specification 163
7.3.1 Normative Organization Programming Language 164
7.3.2 NOPL in practice: the Hospital Scenario 169
7.4 Final remarks on Programming Organizations in Practice 171

8 Organizational Management Infrastructures based on Artifacts 173
8.1 Shaping Organizational Management Infrastructures with A&A . . 173
8.2 Organizational Artifacts 176
8.2.1 Scheme Artifacts 177
8.2.2 Group Artifacts 178
8.3 OMI Execution model 179
8.3.1 Agents using OMI 180
8.3.2 Agents perceiving OMI 181
8.4 Final remarks on Organizational Infrastructures 182

IV Agents, Organizations, Environment: a Unifying Approach 185

9 Embodying Organizations in MAS Work Environments 187
9.1 Situating Agents and Organizations in Artifact Based Work Environments 188
9.2 Environment Management Infrastructures 189
9.2.1 Shaping Environment Management Infrastructures on Organizational Entities 192
9.2.2 Environmental Artifacts 192
9.3 Relating Organizations and Environments 196
9.3.1 Establishing functional relations between organizations and environments 197

9.3.2 Embodied Organization Rules 200
9.4 Final Remarks on Embodying Organizations in MAS 201

10 Programming Embodied Organizations 203
10.1 Embodied Organizations in Practice 203
10.2 Programming Embodied Organization Rules 204
10.2.1 Programming Count-as Rules 204
10.2.2 Programming Enact Rules 207
10.3 Programming Agents in Embodied Organizations 208
10.3.1 Agents at work with Organizational Infrastructure 209
10.3.2 Agents at work with Embodied Organization 212
10.4 From Situated to Embodied Organizations 213
10.4.1 Relevant aspects 213
10.4.2 Limitations and drawbacks 220
10.5 Final Remarks on Programming Embodied Organizations 221

11 Conclusions 223
11.1 Contribution of this Thesis 224
11.2 Future directions 227

A Moise specification for the Hospital Scenario 229

B NOPL specification for the Hospital Scenario 233

Index 242

Bibliography 258

Chapter 1

Introduction

In the last decade, the research in Multi Agent Systems (MAS) has put effort in finding programming models allowing to cope with open systems, characterized by highly dynamic environments, where neither the number, nor the behavior nor the way in which agents interact and access to shared/distributed resources are possibly known at design time. Once the construction of such complex systems is of concern, a multifaceted perspective is needed in order to take into account a series of multiple aspects.

Actual trends seem to account the multiplicity of diverse activities and processes residing in a complex MAS in such a way that the resulting dynamic exhibits some grade of social order, i.e. in terms of evolution of the systems and desired / required equilibrium. For doing this, design models are converging on the notions at the basis of the following first-class elements: *(i)* a set of autonomous entities, namely agents, which are assumed to actively purpose their design objectives; *(ii)* a set of external resources to be exploited by agents to support and fulfill their tasks, namely environments and related facilities; *(iii)* a set of institutional/organizational entities placed for normative regulation, interaction and teamwork management, as those functionalities involving inter agent organizational coordination. Such kind of evolution emphasizes that as MAS research has evolved, the management of certain tasks has been abstracted and gradually shifted from agents to system and infrastructures. As agents can refer to heterogenous models and programming styles, running on different nodes and platforms, they may differ for their architectures, exhibiting different purposes at runtime, with peculiar capabilities to execute activities, interact each other, process and obtain information etc. Besides, environments can be assumed to contain those computational entities that are not autonomous (thus not suitably modellable as

agents), allowing for instance mediated forms of communication or encapsulating relevant functionalities/information to be exploited by individual agents to achieve their desired objectives. Finally, organizational entities are introduced for fastening social dynamics as inter-agent coordination and normative control, and can be involved following different approaches, spanning from electronic institutions, to organizational middleware, coordination models, normative systems, etc.

Whereas recent research seems to account a set of diverse approaches to specifically face with at least one single aspect within the above mentioned, this work aims at proposing a unifying approach where both agents and their organizations are situated in properly designed working environments. Rooted in Agents and Artifacts (A&A) meta-model, which has been recently introduced both in the context of agent oriented software engineering and programming, and fed by a cross-disciplinary ground, spanning from cognitive science to organizational theory and social science, the proposed approach promotes the adoption of artifact based working environments to put in practice a seamless integration between agents from the one side, and organizational and environmental entities on the other side. In this view, working environments take into account both structural and dynamical aspects of the system and are proposed as a programmable infrastructure for describing, specifying and controlling agents in their (social) interactions. Work environments are thus conceived in terms of *workspaces*, representing computational spaces allocated for both enabling and regulating agents activities. Instrumented by artifact based infrastructures and programmable according to situated environmental rules, a workspace provides the computational frame aimed at transparently interceding between the heterogeneous entities inside the system. Besides, being based on laws defining interactions in a unambiguous way, workspaces regulate global dynamics inside the system, for instance in terms of agent environment interactions, propagation of events, functional linking between organizational infrastructures and other resources, and so on.

In this view, artifacts represent the basic building blocks for building either external resources, either organizational functionalities to be exploited as the instruments in the hands of agents to achieve their individual and collective objectives in the context of a problem domain. Thereby, the notion of artifact based infrastructure accounts for considering those situated functionalities that can be dynamically instantiated, shared and used in order to support individual and collective activities. For instance artifacts can be introduced for wrapping resources upon which agents can externalize their activities, but also for mediating and empowering agent interaction and coordination, providing organizational functionalities as situated normative regulation, role management, task allocation etc.

The choice to adopt artifacts as basic element aimed at instrumenting workspaces is envisioned to affect the overall development cycle of the system, thus embedding either a flexible design abstraction, either a suitable programming model (and related technology) needed to smoothly integrate the various aspects of organizations situated in MAS environments. In this view, complex interactions may be shaped obeying either to rules defined locally (i.e. at the level of artifact use/control or within communication between agents) but also to rules which are definable globally (i.e., at workspace level) which are assumed to enact policies and rules aimed, for instance, at regulating the access and possibly the composition of artifact based resources, as well as the the exploitation of social and organizational ones.

1.1 Global view and Objectives

The work described in this thesis draws mainly on research done in the Multi-Agent Systems and in particular on the area concerning either environment and organizational programming. That is, in this thesis we are concerned with the investigation of modeling and programming approaches dedicated to organizational infrastructures situated in MAS work environment.

According to a definition provided by Virginia Dignum, organizations in Multi-Agent Systems (MAS) can be understood as:

> "complex entities where a multitude of agents interact, within a structured environment aiming at some global purpose." [1]

It can be argued that in this definition an organization can be seen as a specific *entity instrumenting the environment where agents interact*. This put some emphasis in relating an organization as an entity inside the agent system which can be inscribed in the context of an *interaction space* which can be summarized as *environment*. The same insight has been recently argumented by Stratulat et al. in [132], according to which both the research lines that in recent years have been addressed at developing organization and environment in MAS can be rooted in the more general approach promoting mediated forms of interactions. The notion of mediated interaction is based on the idea of structuring the interaction space

[1] The definition is provided in the preface of *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, [41].

by adding specific infrastructures responsible to manage complex interactions between many agents[2].

1.1.1 An Integrated approach to Agents, Environment, Organizations

In spite of their common ground, we argue that few attention has been deserved hereto at the challenge to integrate in an unifying view the approaches oriented to environments and organizations. The approach addressed at environment based interactions promotes the idea of environment as first class abstraction, thereby an interaction medium to be modeled and designed as a specific dimension of the system [144]. In this view, a series of aspects have been focused and specifically related to environment design, as for instance how to represent non-agentive entities, how to provide pragmatic actions to agent to work with external entities, how to provide agents with perceptive capabilities, which kind of semantic refers to agent environment interactions and so on. On the other hands, the approaches promoting MAS based on organizations make great progresses by emphasizing the interaction spaces as a social medium. In this view, the interaction space is modeled in terms of social notions, typically detached from concrete computational infrastructures and inspired by human forms of sociality as organizations, roles, groups, norms, etc.

As noticed among others in [146, 132, 34, 18, 91] a scarce attention has been deserved so far in recognizing that many of the current issues in both the research trends could be rooted to a common ground. Accordingly, research promoting environment as abstraction did not grant much attention to organizational and institutional issues, while research on organization centered MAS did not take into account the results coming from environment modeling.

Figure 1.1 provides a global picture of a MAS where agents, environments and infrastructures are jointly present at the same time inside the same system. Besides the representation of the main entities involved (agents, environment, organization) the figure emphasizes the presence links representing relationships between the involved elements: *(i)* A-E (and conversely E-A) links refer to interactions occurring in the (physical) interaction space, as envisaged in environment

[2]Mediated interaction approaches take a different perspective to the one tackled by agent centered interactions, where interactions are established by message exchange between agents with no need of explicit mediating infrastructures (this line is supported by the standardizing works done by the FIPA foundation [52]).

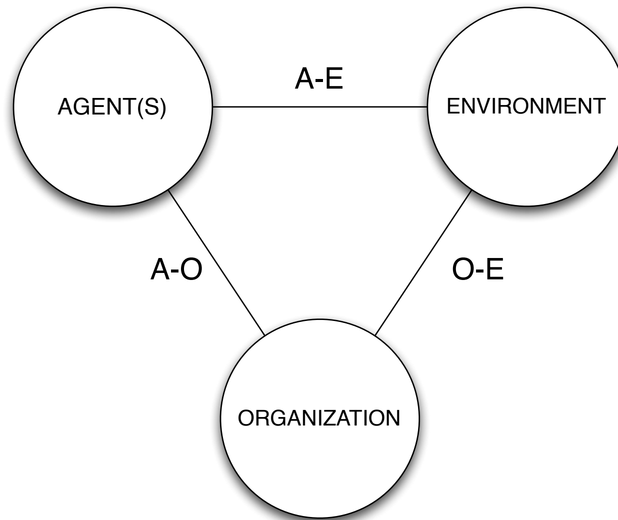

Figure 1.1: Agents, Environments and Organizations, and their bidirectional relationships, inside a Multi-Agent System.

based approaches; *(ii)* A-O (and conversely O-A) links refer to the (social) interaction space as envisaged in organizational based approaches; finally, *(iii)* O-E (and E-O) links refer to the interaction space relating organizational entities and environmental ones.

Among the above mentioned links, we argue in the latter the one which has deserved less attention in research. It figures out the fact that an organizational entity can be fruitfully exploited by agents through a series of environment resources, once these resources could be functionally related to the organization. At the same time, establishing explicit links between an organization installed in a MAS and the environment in which it is immersed, could effectively improve the possibility to situate the whole organization in a concrete context, thus improving the possibilities for monitoring and controlling agent activities with respect to desired state of equilibrium.

1.1.2 Objectives of this Thesis

The starting point of this work is in recognizing that either organizations and environments programming can be rooted in a unifying perspective. This results in

a pivotal role to be played by the infrastructure deployed inside the system aside agents. Accordingly, it rises a series of hypothesis which the next chapters will try to attend. Among these hypothesis, the following ones can be listed as main objectives of the thesis:

- To conceive environments as open systems instrumented by situated infrastructures providing dynamic resources to be exploited by agents to concretely fulfill their pragmatic and epistemic activities.
- To provide a domain-independent programming model for environmental infrastructures as shaped on agent repertoire of actions, thus promoting different interaction styles between agents and situated resources.
- To enable fine-grained management of infrastructures in terms of global specifications, allowing developers to define the rules of interactions and the evolution of the system.
- To realize organizations as situated infrastructures in terms of functionalities to be exploited by agents, and can be shaped on the basis of organizational specifications modifiable by agents as well as system administrators.
- To expand organizational infrastructures to the whole work environment, thus including in a seamless fashion also those environmental supports needed to agents to fulfill their collaborative activities.
- To allow heterogenous agents to seamlessly enter the system, exploit environmental resources and join organizations, with no need to deal with the specific awareness to reason in organizational terms and operate in institutional realities, nor to have multiple capabilities in order to interact with the overall infrastructure.
- To conceive a programming model for organizational entities as embodied resources inside the system, providing a series of distributed functionalities aimed either at enabling agents activities and at regulating their fulfillment—in so doing promoting the achievement of global goals and preventing evolution out of equilibrium.

In fulfilling those objectives in MAS, this work pursues reconciliation of environments with sociality, social interaction with practical interaction, environmental entities with organizational ones, normative regulation with infrastructural support. In order to smoothly integrate the various aspects of complex and situated

organizations a unified approach is proposed, referred as *Embodied Organizations*: It includes either design and programming model, and related technologies, aimed at the definition of distributes infrastructures instrumenting the MAS with *either* environmental and organizational functionalities.

1.2 Overview of the Thesis

To establish at a theoretical level the challenges to provide an unifying view on agents, environment and organizations, the first part of the thesis relates mainly in surveying existing approaches to organizations and environments, focusing in particular on the functional links promoting the view of organizations *situated* in agent environments. The following two parts separately deal with aspects related to environment programming and organizational programming respectively, including the aspects enabling interaction links with agents. In the last part, a unifying approach will be detailed, as addressed at proposing an integrated programming model enabling work environment to be instrumented with situated organizational infrastructures.

1.2.1 Part I · Setting the Stage

Aiming at better situating this thesis in the context of the related work, the first part of the work discusses relevant topics and main research directions on those aspects of MAS research addressed to the area of organizations and environments programming.

In Chapter 2, in particular, the notion of Organization Oriented Programming is described, and a series of existing approaches are analyzed, emphasizing the basic notions and constructs adopted for the specification of an organizational entity. In this view, the $\mathcal{M}$oise approach is introduced in particular as a valuable approach to organization programming integrating several conceptual dimensions in a unique framework. Based on the $\mathcal{M}$oise specification, a derivative programming language based on declarative norms is then described as addressed at governing the overall organizational structure. The resulting programming model will constitute the basis for the approach to organization infrastructures that will be devised in the next part of this work.

Chapter 3 envisages a structured approach to environment programming as a concrete opportunity to overcome some of the actual weakness in situating organization in MAS. Environments – to be considered as the set of external entities

not modellable as agents but aimed at supporting their activities – accounts for a series of theoretical benefits, ranging from augmented interactions, situatedness, monitoring of agents activities, strategies for regimentation/enforcement, and so on. On the other hands, considering environments for these purposes places a series of modeling issues, either on the computational model to be adopted and on the programming approach to be followed to provide a seamless integration with other organizational functionalities. In this view, a series of important aspects are considered to provide a structured approach to organizations situated in computational environments, including data model, action perception models, computational and distribution models, etc. Finally, the Agents & Artifact meta model for MAS (A&A) is introduced as a valuable paradigm for the construction of environments as first class entity of agents worlds.

1.2.2 Part II · Developing Environment Infrastructures based on Artifacts

In the second part of the thesis an environment programming perspective is introduced to define those abstractions, besides agents, which can be adopted to conceive computational model for decentralised, distributed environment in MAS.

Chapter 4 describes a computational model and a related technology environment programming based on the A&A design approach. Both the described design and the programming models envisage *agents* as the basic abstraction which has in charge the autonomous part of the systems, *artifacts* as the abstraction which has in charge the functional part of the system, and *workspaces* as the abstraction providing a concrete locus for defining application domain, grouping together coherently agents and artifacts. Then, CArtAgO is described as the platform providing programming constructs, and run-time support, for building distributed and decentralised work environments based on agents, artifacts and workspaces. Interaction involving agents and artifacts are analyzed taking different perspectives and through the discussion of concrete examples. Basic interactions are enabled since the definition of a basic set of actions extending agents' repertoire with the capabilities needed to operate within CArtAgO. As showed by the end of this chapter, according to the reasoning capabilities owned by agents, the interactions may be conceived also in a more complex fashion, thereby enabling the so called "cognitive" level.

To rigorously define the semantics of agent-artifact interactions and artifact computational behavior, Chapter 5 describes a formal model of artifact-based en-

vironments. The chapter defines the entities involved in a MAS based on agents, artifacts and workspaces in terms of their configurations. Then it focuses in particular on the dynamics occurring inside a single workspace, involving in particular interactions between agents and artifacts, and provides a formal description using operational semantics. Event based mechanisms regulating the evolution of the workspace deserve in this case a particular attention, being introduced in order to consider further intra-worksace dynamics characterizing the system.

Indeed, after having discussed micro-level interactions occurring between agents and artifacts, in Chapter 6 a complementary mechanism for specifying global dynamics inside the workspace is provided based on events. The mechanism involves the definition of event driven rules governing primer mechanisms inside a workspace and is aimed at providing a further, pervasive management for the work environment. Thereof, environment dynamics can be programmed not only on the basis of the behavior of the single agents interacting with artifacts and workspaces, but also at a macro level, i.e., defining laws governing global workspace dynamics. This is done by introducing general rules, triggered by environment events, allowing to manipulate the space of events and affect workspace configuration, i.e. triggering operation on artifacts, creating and disposing artifacts, enabling and disabling operations, and so on.

1.2.3 Part III · Developing Organizational Infrastructures based on Artifacts

The third part of the thesis introduces an organization programming perspective and draws the basis to define specific artifact based infrastructures inside the MAS to be exploited by agents for organizational purposes. Organizational entities devise their specification along multiple dimensions aimed at regulating the behavior of complex societies of agents and at promoting the fulfillment of complex tasks in which groups of agents cooperate in order to achieve shared goals. The organizational entities will be then reified inside the work environment in terms of artifact based infrastructures, so as to be suitably exploited by agents in order to support either individual and collective activities.

In particular, Chapter 7 envisages a programming model defining an organizational entity to be deployed inside the MAS as a concrete framework which agents can interact to in order to exploit organizational functionalities. On the basis of the well suited $\mathcal{M}$oise organizational modeling language, a concrete example of organization will be sketched, according to a specification given along different

conceptual dimensions. According to the specification of a concrete use case the programming model for a concrete organizational entity is provided, including the normative programming language addressed at regulating the overall organizational dynamics in concrete organizational infrastructures to be deployed inside the work environment.

Once the general structures involved in a organizational entity have been analyzed, Chapter 8 addresses the problem to build and deploy an organizational infrastructure in practice. The approach adopted in this chapter is based on A&A model and makes use of artifacts as the main abstraction tool to implement the whole organizational infrastructure. Organizational artifacts are thus deployed in the work environment with the aim to instrument workspaces with organizational facilities, thus reifying and modularizing the functional part of the organizational entity. The result is a decentralised – and configurable – layer of organizational resources, that can be either perceived and used by agents as first-class entities of their work environment. Thereby, the enabling and governing function inherited by artifacts allows the infrastructure to effectively enact norms inside the system. Through an internal normative model providing the infrastructures with a flexible mechanism to dynamically manage institutional states, norms are first monitored during their life cycle, and then applied according to regimentation or enforcement policies.

1.2.4 Part IV · Agents, Organizations, Environment: a Unifying Approach

The last part of the thesis is addressed at proposing a *seamless* integration between organizations, agents and their work environment. This part starts by considering that the sole adoption of artifacts based infrastructures to reify organizations inside MAS constrains agents to be aware of complex structures and constructs proper of organizational specification: agents must know and manipulate low level primitives related to group, roles, mission and norms which may be not proper of an application domain. Besides, as far as organizational infrastructures have been modeled, a weak support is given for monitoring and controlling – at the organizational level – other environment infrastructures deployed inside the same work environment.

To bridge this gap, in Chapter 9 the notion of embodied organization is introduced as a programming model aimed at unifying organizational and environmental infrastructures. It includes a programmable layer for specifying functional

links between organizational entities and environmental ones. In doing so, the possibility to conceive environment infrastructures aimed at better situating the organizational entities inside work environments is sketched. The envisaged computational model marks a clear separation of concerns between the organizational and the other environmental infrastructures. At the same time, by introducing the possibility to explicitly specify constitutive rules based on events, the model enables functional relations between the two systems. Finally the approach promotes multiple interaction styles between organizations agents and their environment.

After having detailed an abstract model for Embodied Organizations, Chapter 10 details a concrete description on how the approach to can be engineered in practice. Adopting the scenario previously introduced as guideline, the chapter is enriched by a series of concrete examples aimed at providing a practical methodology to developers in programming an embodied organization. A final section analyzes strength and weakness of the proposed approach, discussing the main features with respect to the devised challenges.

The thesis concludes in Chapter 11 by listing the contributions of this work and by providing final discussions and future research directions.

1.3 Relevant Issues we do not address

A unifying approach in programming MAS would require to cope several aspects characterizing a complex software system, typically inhabited by many heterogenous entities and infrastructures which dynamics and evolution could be not understood completely at design time. This work mainly faces with proposing integrated organizational and environmental infrastructures in MAS, leaving aside, actually, other issues that are not centered on these main aspects. In particular, we do not address in this work aspects related to the agent computational models, referring when possible, to existing works already presenting a complete semantics of agent reasoning (among others [14, 31, 148]). This choice could be argued with the need to guarantee the whole system as much open as possible with respect to agent architectures and models. In addition, as remarked by Ferber and Gutknecht [49] in their seminal work on organization centered MAS, an organizational entity should make no assumption on agent models. We also do not provide insights on specific capabilities required by agents to reason in organizational terms, i.e. by taking into account explicitly organizational notions within their reasoning phases. On the other hands, one of the outcomes of the proposed approach is exactly in enabling not aware agents to fruitful interact with organizational entities without

taking into account complex organizational constructs.

By taking an approach oriented to a mediated kind of coordination (where the mediation role is played by artifacts), another aspect that we leave aside is the dialogical dimension of interactions, namely explicit message based communication between agents and, possibly, between agents and the infrastructures situated inside the work environment. In this case we actually leave aside the integration of message based interactions within the sphere of influence of the organizational entities. Once needed, we will refer to already existing mechanisms as the ones based on FIPA-ACL standards already implemented inside the adopted agent platforms.

Part I

Setting the Stage

Chapter 2

Organizations in MAS: Theories, Scopes and Directions

This part of the thesis provides relevant topics and main research directions on those aspects of MAS research addressed at organizations and environments programming.

In this chapter, in particular, the issues related to programming organization are discussed aiming at better situating the following of the thesis in the context of the related work. The notion of Organization Oriented Programming is described, and, taking a programming perspective, a series of existing approaches are analyzed, emphasizing the basic notions and the programming constructs adopted for the specification of an organizational entity. In particular, the $\mathcal{M}$oise approach is introduced as an applicable approach to organization programming integrating several conceptual dimensions in a unique framework. Based on the $\mathcal{M}$oise specification, a derivative programming language based on declarative norms is then described as addressed at governing the overall organizational structure once deployed inside the MAS. The resulting programming model will constitute the basis for the approach to organization infrastructures that will be devised in the next part of this work.

2.1 Organizations in MAS

The definition of a proper organization for a MAS is not an easy task. On the one hand the organization can be too flexible, and then it does not help the achievement of the global purpose. On the other hand, it can be too rigid, thus affecting

agent autonomy and removing any perspective in terms of possible evolutionary dynamics. In the last years, several approaches to build organizations have been proposed in literature, each characterized by different perspectives on MAS, using different primitives and mechanisms addressed at providing organizational functionalities, and adopting different strategies for monitoring and controlling interactions and activities performed by agents inside the system. Such different approaches originated heterogeneous organizational implementations, addressed each at facing with particular challenges, requirements, application domains.

The first part of this chapter resumes background works on those aspects of MAS research addressed at organizations programming, with the aim to better situate the following of this thesis in the context of the related research. In Section 2.2 the description is structured according to the main aspects and challenges that, in recent years, characterized the evolution of organization programming in MAS.

The second part of this chapter focuses on $\mathcal{M}$oise, an approach to organizational programming based on specification of separate dimensions each dealing with different aspects of a complex organization (Section 2.3). A $\mathcal{M}$oise specification is interpreted and executed by an organizational entity described in Subsection 2.3.1, that is deployed in order to instrument the MAS with organizational services and functionalities exploitable by agents. As far as the organizational entity will be conceived based on $\mathcal{M}$oise, its functioning will concern processing events like agents entering and/or leaving it, group creation, role adoption, goal achievements, mission commitments, etc. Whereas agents concurrently operate in the system, i.e. interacting each other and with the infrastructures already available in order to meet their design objectives, the organizational entity has in charge the task to maintain a coherent state of the organization. The proposed model assumes organizations storing institutional states and managing explicit norms, to which agent are forced to obey by the mean of constraints imposed at runtime by the system (Subsection 2.3.2). Norms, in particular, are introduced as a regulatory mechanism, thus allowing the application of behavioral constraints to agents in terms of conditional obligation and permissions. The basic concepts at the basis of the normative specification to be handled inside the $\mathcal{M}$oise based organization are resumed in Subsection 2.3.3, including mechanisms for norms management and application. Finally, the chapter concludes in Section 2.4 with final discussion and remarks.

2.2 Organization Oriented Programming

The complex requirements of wired and distributed software systems, as well as the increasing computational power of hardware platforms are originating a growing interest towards organizational approaches in programming complex and distributed software systems (see, among others, [59]). The use of organizational and normative concepts as inspired to human societies is widely adopted as a suitable modeling approach in complex system development, while recent research lines in Multi Agent Systems (MAS) area originated many proposals in this direction (for comprehensive overview see [41]).

A first research line in MAS area investigates organizations from an agent centered perspective, where the focus is on the particular capabilities needed for agents in order to comply with an organization [25, 22]. The perspective is straightforwardly rooted in typical agent oriented modeling and programming. A main challenge in this view is in conceiving the required capabilities for *organization-aware* agents, namely capabilities enabling agents to reason in organizational terms, i.e. by taking the organizational specification into account during decision making [137]. In this view, the practical reasoning is amenable of changes once agents are concerned with norms to fulfill besides goals to achieve [83], or with deliberating whether enacting and deacting roles [43, 32]. A remarkable line of research takes the agent centered perspective to emphasize the emergence of cognitive phenomena as norm formation inside societies of intelligent agents. For instance, Sen and Airiau investigate norms as a self-enforcing mechanism which may evolve in a bottom-up manner [129]. Emergence of norms is also at the basis of the model proposed by Rosaria Conte et al., given the extent that norms embed and "immerge" in agents' mental attitudes [4].

Other research lines devised the notion of organization centered MAS, where a strong separation of concerns is typically marked between the organizational specification and the agent models. In this perspective the adoption of explicit organizational models allows the balance of global organizational requirements with the autonomy of individual agents. In turns, the organization oriented approach provides the possibility to express and make explicit one or more patterns of behavior which are imposed in a top-down fashion to the agents [50, 11, 150]. The organization has, in this case, the aim to constrain the evolution of the system by ruling over agent activities and interactions towards the fulfillment some desired state of equilibrium, or towards the achievement of some global objective. Such an approaches have been adopted in different multi-agent methodologies to create organizational patterns by design (among others [49, 150, 48, 72, 138, 67]).

Besides, they are also adopted at a programming level, to allow agents to interact at an application level with the organization to which they take part as an explicit entity deployed inside their system.

A recent trend in MAS has been addressed at Organization Oriented Programming, aimed to provide languages that both the system developer and the agent themselves (as in the case of self-organization) can use to specify a program that defines the organizational / institutional dimensions. In this line, the definition of specific *Organization Modeling Languages*, as it has been provided by Boissier [12], has been adopted to refer to those programming languages specifically conceived to design and specify organizations. An OML specification is thus a program, typically collecting and expressing specific constraints and cooperation patterns that the organization designer assume to be applied to the agents participating the organization functioning. Typically, the reification of OML specifications originates *Organizational Entities* (referred as OE), assumed to conceive those multifaceted constraints and rules expressed in OML so to provide a set of services and functionalities needed by agents to exploit concrete organizations inside the MAS. The deployment of the organization is finally achieved by installing concrete architectures and modules implementing the organizational entities inside the system: at this level the organization can be viewed as an *Organizational Management Infrastructure* (OMI) and is assumed to embody the organizational services in concrete computational components exploitable at an application level.

2.2.1 Agent, Groups, Roles

The way in which organizational entities are developed in MAS borrows concepts typically referred to societies and organizations of human beings. The concept of role, before than others, has deserved main attention being one of the cornerstones in developing systems based on the social metaphor of organizations. Roles have been applied either to design modeling and methodologies [79, 150, 90] either to programming languages, frameworks [7] and technologies [126].

One of the first attempts to provide a systematic characterization of MAS organizations, including roles, is due to Jaques Ferber and Olivier Gutknecht [49], according to which an Organization Centered MAS (OCMAS) is not conceived in terms of agentive mental attitudes (as opposed to Agent Centered MAS), but only on capabilities, constraints and organizational concepts like groups, roles, positions, communities, collective goals, functions, etc. The concepts which have been placed at the basis of organizational characterization by Ferber and Gutknecht can be still found in many of the current approaches: *(i)* an organization is consti-

tuted by agents (individuals) that manifest an autonomous behavior; *(ii)* the overall organization may be partitioned into smaller set of agents (groups) that may overlap; *(iii)* agents' behaviors are functionally related to the overall organization through the concept of role; *(iv)* agents are engaged in dynamic relationships which may be typed using taxonomies of roles, tasks, protocols, thus describing a kind of supra-individuality; *(v)* types of behavior are related through relationships between roles, tasks and protocols.

Following these basic principles, Ferber et al. presented the Agent Group Role (AGR) modeling approach, that, in the context of organizational programming, explicitly introduced three basic primitives, that are structurally connected and not further reducible to other primitives:

Agents Agents are the autonomous, (pro) active entities inside the system. Agents are aimed at achieving their design objectives, and for doing this they may play roles inside groups. An agent may hold multiple roles, being member of different groups at the same time. In order to make the organization as open as possible, no assumptions should be made on the particular model and architecture employed by agents.

Group Groups define the structural aspect of an organization, and are conceived as sets of agents sharing common characteristics. A group is viewed as a context for a pattern of activities and is used for partitioning the global organization into smaller organizations. In AGR, groups also define the communication channels between agents, while agents may concurrently belong to several groups.

Role A role is viewed as an abstract representation of a functional position which an agent can play inside a group. A role describes the behavior of agents in abstract terms, defining what the organization expects form agents playing it, and, in AGR, also the responsibilities (requirements, obligations, skills) deemed for agents to play that role. Roles belongs to groups, and to stay in a group an agent has to play a role in it. The same role can be played by different agents at the same time.

These basic notions are the basic building blocks of the AGR approach to OML, and have been placed at the basis of the MadKit OMI, provided as a framework for programming AGR-based organizations [50]. MadKit implements the organizational entity inside the MAS as a core layer of basic functionalities to let agents join groups, associate roles to agents and enable interactions only for member of the same group [57].

2.2.2 Collective Intentions

Although the AGR model specifies what kind of behavior is expected by agents in the context of an organization, it says nothing on *how* such behavior has to be performed in practice. This aspects is covered by other OML approaches, as for instance the one given by Shell in the TEAMwork model (STEAM, [134]). In this case the behavioral aspect of agents is the pivotal one in the OML specification: exploiting the notion of joint intentions and collective goals, the STEAM approach enables team of heterogenous agents to coordinate their activities by the mean of shared plans specifying how goals could be jointly achieved, i.e. in terms of courses of actions and workflow to be performed by agents. In doing so, the STEAM approach promotes synchronization and coordination among agent teamworks, for instance using mechanisms establishing communication between agents aimed at improving decision making, and allowing re-assignment of roles once critical situations arise.

AGR and STEAM models focus on complementary aspects, the former being centered on social and structural aspects of an organization, the latter being centered on behavioral and coordination aspects.

2.2.3 Social Laws

Besides roles, groups and shared plans, a well suited paradigm for regulating complex systems has been rooted to the use of explicit social laws aimed at coordinating individual agents and rule the (emergent) behavior of the society as a whole [131]. A further step has been proposed in enriching organizational specifications provided by the OMLs with the introduction of explicit norms [23, 46, 9, 42]. In this view a norm states a series of obligations, permissions and prohibitions, namely those behavioral constraints that agents should obey during their activities inside the organization. An important aspect is that norms compliance has to not affect agent autonomy: agents should remain able to read, to represent, and to bring about the organization so to autonomously decide whether to oblige the norms prescribed by the organization or not. This aspect in particular requires that the agents understand an organizational specification and are able to translate it in concrete actions complying the ongoing norms. A typical scenario in this view is about agents who need to balance the benefits of violating norms against possible negative consequences resulting from this.

2.2.4 Electronic Institutions

Among the organizational aspects introduced by existing approaches, a relevant one is related to the dialogic dimension of interactions, namely the regulation of the communicative acts performed by the agents. Massage based communication and Agent Communication Languages (ACL) are an important approach (probably the main one) to agent interaction [52]. The dialogic dimension is otherwise the pivotal one in the approaches based on Electronic Institutions [48]. Electronic Institutions infrastructures are specified according to the AMELIE OML [47] and can be modeled using the ISLANDER graphical tool [45].

In AMELIE, an institution is viewed as a dialogic system where the only admitted interactions are speech acts that agents exchange. Which speech acts the agents can perform and which roles the agents can play inside the organization are defined by the dialogic framework according to the OML specification. The interactions amongst agents take place in so-called scenes, which are assumed to model situated task environments as workplaces (localities) where agents can exchange messages in order to fulfill their objectives. Communication protocols specifying the allowed messages are associated to the scenes. Messages are formed in terms of illocutions, and are based on shared ontologies to guarantee a common understanding of their contents. In AMELIE, the protocols are also normative, namely they specify which speech acts can be uttered by the agents interacting in each scene. How agents can move from scene to scene respective the normative dimension is specified by the performative structure, that is a network capturing the transitions that agents can make from scene to scene. Finally, certain circumstances in one scene (e.g., winning an auction) might lead to obligations in other scenes (e.g., an obligation to pay). In this case norms are outside the scope of scenes, and are expressed as global norms which scope is the whole institution. These norms specify which obligations hold when certain speech acts have (or have not) been uttered in particular contexts. Just like the norms expressed by the protocols and the performative structure, these obligations can only refer to speech acts.

A particular functionality inside the Electronic Institutions is provided by the so called governor entities, which are assumed as proxies mediating message exchange. Governors both rule and enable the dialogical interactions, in so doing taking a role which resembles the one played by artifacts in A&A approach which will be introduced in the next chapter. In addition, governors are the controllers for the respect of specified protocols and norms. Besides intercepting messages exchanged between agents, governors are, also responsible to mediate A-O inter-

actions as depicted in the reference Figure 1.1.

The protocol based interactions provided in EI are at the basis of a series of application integrating the AMELIE with already existing open standards, as the ones introduced by the web services stack and SOA [149]. Among others, [39] proposed a composition of (web) services in a open MAS as regulated by EI. The mediation between agents, EI and services is in this case performed by the additional figure of Institution Middle Agent (IMA), which has been assumed as provider of composed services. To serve the requests raised by agents, the IMA exploit available services as yellow pages, directory facilitators, match-makers in respect to the protocols defined inside the EI.

2.2.5 Normative Systems

In the case of open system, the transitions occurring in complex agent formations can not be envisaged by the developer at design time. The requirements of openness according to which few assumptions can be made about the behavior, the purposes and the internal model of agents working inside the MAS, has raised the need to deal with explicit normative specifications to be handled at a programming level. Programming norms is thus aimed at regulating the evolution of the system and at coordinating the whole system within desired states of equilibrium. In this view, a series of additional charges are assigned to the organizational entities namely, to *monitor* agents to conform the constraints specified by the norms, to *ensure* agents to comply with norms and their specified prescriptions, to support at runtime the possibility for the norms to change dynamically their prescriptions.

Monitoring Norms Compliance

Two main approaches are followed in literature for monitoring norm compliance, the former assuming the infrastructure to automatically relieve norm violations, the latter adopting special organizational agents in the role of controllers. The first approach is followed among others by Dastani et al. [34, 135], where the normative system is equipped with a set of infrastructures allowing an automatic detection of norms violation. The second approach, focuses instead on the special role played by agents in detecting and judging norms violation. Among others, in [2] sanctioning and repairing activities addressed to agents violating norms in Electronic Institutions have been specified in terms of speech acts that should be uttered by so-called *enforcer* agents; Similarly, in the context of Electronic Institutions, *staff* agents have been proposed in Campos et al. [18], which aim is to

improve the monitoring activities of the institutions in the context of applications situated in real environments. Special *institutional* agents have been proposed to promote autonomic control, based on detecting and preventing norms violations [15]. Also in this approach, staff agents are assumed to play institutional roles while external agents participating in the institution are assumed to apply to roles as external users.

Ensuring Norms Compliance

After having established mechanisms aimed at verifying norms compliance, an additional charge for OEs is to *ensure* agents to oblige norms and their specified prescriptions. The application of the norms, and the management of norm violations, are typically achieved by enforcement and regimentation strategies, the former being based on some sanctioning policy to be applied to agents violating the norm, the latter being based on preventing violations by the mean of access control to resources [71].

Regimentation is a mechanism that simply prevents the agents to perform actions that are specified as forbidden by a norm. That is the *barrier effect* to which Castelfranchi refers in [23]: given a regimentation of a norm, agents are physically unable to fulfill a given action because of practical environment constraints. For instance, in physical environments obstacles could prevent the fulfillment of a given action, or, in computational systems, a given action can be disabled by a security mechanism. Regimentation of actions is aimed at preserving important features of the organization as invariant properties.

Enforcement While regimentation is a preventive mechanism, enforcement is a reactive one. Enforcement is applied “a posteriori”, namely after a possible norm violation occurs. Enforcement has the prerequisite that agents have the choice to obey or not to the norm according to their local view of the organization. On the other hands, the fulfillment / unfulfillment of the norms should be detected and evaluated as a possible violation, and then judged as worth of sanction / reward or not.

It is worth remarking that these two mechanisms allow to balance between constraining pivotal properties of the system without affecting agents’ autonomy. This is of particular importance to let the system to self-organize and evolve as

in the case of open systems, where heterogeneous agents are supposed to enter and leave with unknown architectures and purposes.

The norms can be supported either as regimentation or enforcement mechanisms depending on which side the designer wants to give more emphasis with respect to the application domain. Basically, regimentation can be assumed to fully constrain the actions of the agents, while enforcement is adopted when some violation is allowed (or even desired). Enabling both regimentation and enforcement is a feature introduced, among others, in ORA4MAS [63], and will be described in the last part of this book as a pivotal characteristic of the proposed organizational model.

Dealing with Norms Dynamics

An additional aspect in norms management is related to the possibility to adapt the normative specification at runtime. In this view, a couple of works have been recently proposed in order to extend the framework of Electronic Institutions with an improved norm management system. Besides presenting both regimentation and enforcement mechanisms for norms application, the works presented by Bou et al. [15] and Campos et al. [18] are addressed at introducing additional autonomic capabilities to allow a dynamical adaptation to changing circumstances through runtime management of norms. In this view, organization aware agents – called institutional agents – are introduced to regulate a given population of external agents by detecting and also preventing norm violations. The approaches are based on the tuning of a transition function, which in turn is assumed to specify how norms may evolve based on a series of institutional objectives which are matched with properties and facts observed inside the application domain. In doing this, two different mechanisms are envisaged: while in Bou et al. the approach deals with using on-line learning mechanisms to adapt the transition function to fulfill the institutional objectives (i.e. with respect to desired equilibrium), in Campos et al. the rules for adapting norms are defined off-line by the developer.

A different perspective is taken by approaches defining the multifaceted dimensions of an organizations on the basis of explicit languages based on norms. In this view, an organization results specified in terms of norms, even if originally conceived at an higher level of abstraction within an OML facing multiple dimensions. Such an approach is followed, for instance, by two recent works on ISLANDER aimed at defining mechanisms allowing an automatic translation of the complex normative aspects of Electronic Institutions into simplified constructs interpreted by rule based engines [29, 53].

The idea to manage norms through a rule based engine is followed also by Aldewereld et al. [3], where the management of declarative norms takes into account an explicit relationship through three different dimensions of the system, thereby distinguishing between the normative domain, the organizational domain and the concrete (environmental) domain. The approach in particular implement a mechanism – based on conditional constitutive rules – for regulating norm application in those situations in which both organizational and normative context can change over time, due to changed conditions in the concrete level. In this view, making all three levels and the links between them explicit is beneficial for dynamic domains and enhances the ability of agents to reason about context and norm changes.

Instead of translating norms into rules, other approaches have introduced specific programming languages dealing with norms as an explicit construct. In this line, the work by Dastani, Tinnemeier et al. [34, 135] proposed a programming language using declarative norms in terms of conditional obligations and prohibitions. Norms are specified through declarative rules, where the antecedent range over concrete state of the system and the consequent specifies enforcement mechanisms to be handled once the norm is applied. Differently from other approaches, this programming model explicitly accounts the possibility to refer norms to the concrete states of the environment (as opposed to "procedural" norms only referring to actions to be performed by agents)[1].

An approach to automatically shifting from an abstract OMLs to a simplified language based only on declarative fluents has been recently proposed by Hübner et al. [61, 62]. They introduced an automatic translation of a $\mathcal{M}$oise OML specification into a Normative Programming Language (NPL) which can be used to represent institutional facts, rules and norms actually regulating an organizational infrastructure. This work, in particular, will be detailed in the next section, being both the $\mathcal{M}$oise and related NPL the reference programming languages adopted to define the organizational specification proposed in this thesis (see also Chapter 7 for the discussion of the $\mathcal{M}$oise approach applied to a concrete case study).

---

[1] The aspects related to the support of concrete states as opposed to normative states will be better analyzed in the next chapter, where dealing with the environment dimension will be addressed as a specific challenge of our organization programming approach.

2.3 Moise: an organizational model based on structural, functional and deontic dimensions

The Moise OML and the ORA4MAS OMI are at the basis of the approach to organizational programming proposed in this work. They will be better detailed and extended in Chapter 7, Chapter 8 and Chapter 9 where a unified programming model will be devised in order to concieve concrete organization as fully embodied infrastructures situated in agents work environments.

Moise has been developed in order to provide a comprehensive approach to OML, integrating roles, groups, norms and collective goals. It improves the original Model of Organizations for multI-agent SystEms (referred as Moise$^+$ [65, 64]) which identified three separate dimensions upon which an OML can be structured.

Functional dimension The *functional dimension* specifies how a set of collective goals should be achieved, and concerns the global functioning of the organization, namely: the specification of global plans, the policies to allocate tasks to agents (*missions*), the coordination of plans execution, and the quality (time consumption, resources usage etc.) of a plan. The decomposition of global goals results in a set of goal-trees, called *functional schemes*, where the leafs are sub goals that can be individually achieved by the agents. This dimension has then been associated to a sort of memory for the organization, where the best practices to achieve collective goals can be stored (e.g., similarly to TAEMS [113], STEAM [134]).

Structural dimension The *structural dimension* addresses a more static aspect of the organization: it refers to the structure, namely, the groups, the roles inside groups, the relations among roles, etc. In each group specified along the structural dimension, the global purpose is accomplished while the agents have to follow the obligations and permissions that their roles entitle them (e.g., similarly to the AGR model [49]).

Deontic dimension Using an approach similar to ISLANDER [48], OPERA [40], the *deontic dimension* focuses on the definition of high level norms that the agents should obey. Specific norms are provided here in order to bind the structural dimension with the functional one. This is achieved by indicating, for each role, which are the permissions and obligations with respect of the envisaged missions.

The idea at the basis of $\mathcal{M}$oise$^+$ was that an initial adequate organization is normally set up by the MAS designer, however this may become not suitable in complex systems where nor the architectures of the participating agents, nor the multi-faceted patterns of cooperation may be known at design time. Thus, from a developer perspective, defining an organization focusing from time to time on clearly separate aspects is of course an advantage in terms of flexibility and maintenance of the overall system. From an agent point of view, having three inspectable dimensions to work with means having more information to reason about the others positions inside the organization. If the organization model specifies separate dimensions while maintaining suitable independence, then the participating agents can be more effective in adapting their behavior to organizational schemes. Besides, this allows to improve interaction inside the organization, thus better eliciting cooperation among single agents.

Making explicit the functional dimension eases the computational load for agents by providing suitable collective plans – to which agents may commit through missions – every time they want to coordinate each other. Even with a small search space in the problem domain, this may hold to an hard problem for agent decision making, given the fact that the fan out of possible commitments for an agent grows with respect to the amount of other agents in the group. An additional motivation to explicitly introduce the functional dimension is the possibility to store plans in the organizational memory, as a sort of social intelligence.

The $\mathcal{M}$oise$^+$ approach provides the chance for agents to adapt and change the organization in a bottom-up process, for instance installing and or updating new pattern/structures at runtime. Such a feature corresponds to the combination of agent-centered MAS organizations and organization-centered approaches.

In order to improve the normative aspects inside the OML, $\mathcal{M}$oise^{INST} [10] has been recently proposed as an extension of $\mathcal{M}$oise$^+$. The $\mathcal{M}$oise^{INST} OML introduces an renewed deontic dimension aimed at better specify norms, including timely deadlines for obligations and permissions. Besides, it adds to the OML the so called contextual dimension, aimed at including the specification of the possible states in which an organization may operate. This dimension explicitly accounts for situated conditions, given for instance by changing contexts inside the application domain. The $\mathcal{M}$oise^{INST} OML also considers organizations in dynamic environments, providing a specific support to adjust organizational structures on the basis of the changing contexts.

2.3.1 Organizational Entities

Organizational entities provides the proper abstraction tools, a related technology and a runtime support of the execution of large-scale, distributed and open MAS where organizational entities are explicitly defined. In general, an OE can be seen as a particular system inside the MAS, that is deployed to instrument the interaction space in order to provide organizational support to agents. In this view, an OE provides organizational facilities at an application level, making it available one or more organizational functionalities exploitable by the participating agents. An OE is thus responsible for interpreting an organizational program, possibly based on notions specified by the system developer at an high level of abstraction. Accordingly, it is responsible for the runtime management of the organization with respect to the specifications expressed by the organizational program.

At an application level, the $\mathcal{M}$oise$^+$ OML is managed by the $\mathcal{S}$-$\mathcal{M}$oise$^+$ system, an OE combining the structural, functional and deontic dimension in an organizational middleware exploitable by heterogenous agents [66]. $\mathcal{S}$-$\mathcal{M}$oise$^+$ is realized as a series of middleware components, ensuring the coherence of the organization as agents playing the right roles, and forcing global properties as well-formedness of groups. Besides, the same system provides organizational functionalities to agents, which are enabled to interact with the OE by the use of basic actions as adopting roles, commit to missions, join/leave groups and so on. In this case, the middleware is assumed to inform agents by signalling events once relevant and applicable goals can be pursued and regiment the commitment to mission for which agents have no permissions to act. The interfaces provided in $\mathcal{S}$-$\mathcal{M}$oise$^+$ as well as a properly conceived event based signalling, establish, at a programming level, the interaction model between organizational entities and agents. In particular $\mathcal{S}$-$\mathcal{M}$oise$^+$ enables A-O relationships (see the reference Figure 1.1) through a mechanism based on a dedicated component called “Orgbox”, to which the agent sends requests for actions that it would like to execute to the organizational entity, which decides whether the specific action can be finalized or not.

To improve the programming support allowing agents to work with organizational entities, $\mathcal{J}$-$\mathcal{M}$oise$^+$ was proposed as an integration layer enabling *Jason* agents to seamlessly work in a $\mathcal{M}$oise$^+$ middleware. In this case, the relationships between agents and organization (A-O relationship in Figure 1.1) is entirely based on an augmented action repertoire, by which *Jason* agents are enabled to effortlessly use middleware services in a transparent fashion. Besides, the agent perception model is entirely conceived in terms of events which are dispatched by the middleware. This allows agents to fully exploit their reactive abilities, since

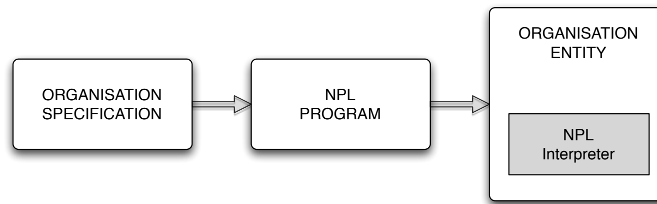

Figure 2.1: In order to simplify the internal functioning of an Organizational Entity inside the MAS, the specification given in Moise-OML is translated to a Normative Programming Language (NPL).

events are translated in percepts which are native events for agents and can be handled accordingly.

An important aspect to remark here is that both $\mathcal{S}$-Moise$^+$ and $\mathcal{J}$-Moise$^+$ lack monitoring mechanisms allowing the OE to be automatically notified about the ongoing changes in the application domain. In other terms, each change occurring inside the system has to be notified to the organization by the mean of agent actions aimed at informing and updating the organizational configuration. For instance, when an agent fulfills a goal belonging to its obliged missions, it is on agent's own responsibility to inform the middleware about the achieved goal. As it has been noticed among others in [34, 137], this is a strong assumption on agents responsibilities, as it assumes agents that strongly agreed on organizational specifications and are somewhat responsible to perform an additional set of activities besides their purposive behavior, so as to inform the organization.

By taking into account an organizational specification shaped on multiple dimensions, we may envisage that the practical implementation of an OE requires the continuous management of complex organizational entities, each based on several heterogeneous concepts. This aspect is particularly complex for organizational models that consider elements with heterogeneous nature like groups, roles, common goals, norms, etc. Typically, each of these notions has its own life cycle inside an organization, they are connected together and are constrained by many properties (e.g. well-formedness, role compatibilities, links, cardinalities, etc.) that need to be concurrently satisfied to keep the overall organizational in a consistent state.

As described in the previous sections, many of the organizational concepts in Moise$^+$ have an high degree of coupling with the other ones. For instance, structural and the functional dimensions must be related from time to time with obli-

gations and permissions related to the deontic specification. $\mathcal{M}$oise^{+} allows the specification of explicit norms, that are always related to roles and goals through obligations (or permissions) for agents to commit to missions. In so doing, the designer can define norms such as, for instance:

> *Norm* : *An agent playing the role "doctor" is obliged to visit another agent playing the role "patient" if another agent playing role "visitor" (or the "patient" itself) has already booked the visit.*

In this case, the OE is responsible for identifying the activation of the obligation to commit to the involved missions either for the doctor either for the patient and the visitor. Accordingly, the system has also to ensure the compliance to the norms for the agents playing the corresponding roles. In this perspective, as one may argue, the more complex is an organizational specification, the more heavy is the computational load in charge of the related OE. By taking into account the above mentioned issues, in the next section a modeling approach addressed at simplifying the computational model of organizational entities will be described. In particular, the approach devises the model for an organizational entity built around the notion of organizational states, rules and norms, where norms are assumed as the basic constructs at the basis of a programming language aimed at specifying the overall organizational dynamics.

2.3.2 Recasting Organizational Entities as Normative Systems

Based on the improved notion of norms already introduced in $\mathcal{M}$oise^{INST} [10], Hübner et al. recently proposed a unifying approach – simply referred $\mathcal{M}$oise – improving the $\mathcal{M}$oise^{+} OML based on the notion of declarative norms [61, 62]. On the basis of the resulting $\mathcal{M}$oise specification, the proposed model is then assumed to translate the OML specification to a simpler one, based on the sole use of norms. In this approach, the organization turns to result as a normative system, namely the internal functioning of the OE is mainly concerned with providing a coherent mechanism capable to interpret and manage norms at runtime.

The approach assumes a normative programming language to be fed as the input specification of the OE. This allows to re-cast the OE configuration by translating an abstract organizational model into a more simple normative model. Accordingly, it envisages the decomposition of the OE into smaller entities which are *(i)* a translation component which fulfills the encoding from an organization modeling language to a normative programming language and *(ii)* an interpreter for the normative language to be run inside the OE. The OE is then in charge for

interpreting and managing the norms expressed by the normative programming language with an internal engine. Normative constraints and rules are used by the OE to establish its organization state and to govern the organization dynamics, i.e. with respect to the agents activities (see Figure 2.1). In the next section, the basic constructs at the basis of normative language are described.

2.3.3 Managing Organizations with a Normative Programming Language

As said, the approach followed to model an OE assumes to recast the organizational specification, placed in abstract terms, into a lower level language, based on normative constructs. The approach assumes, in particular, a starting specification given in Moise Organization Modeling Language, and a target language based on norms referred as Normative Programming Language (NPL). The basic construct of NPL is the norm, that has been specified in [61] as a general rule of the type:

$$\texttt{norm id:}\quad \varphi \texttt{ -> } \psi.$$

where `id` is a unique identifier of the norm; φ is a formula that determines activation condition for the norm; and ψ is an expression declaring the consequences of norm activation. Two types of consequences of norm activation are considered:

- $fail - \texttt{fail}(r)$: represents the case where the norm is regimented, namely every tentative to reach the state expressed by the formula φ will fail. Argument r represents the reason for the failure;
- $obl - \texttt{obligation}(a,r,g,d)$: represents the case where a new obligation is created for some agent a as the consequence of the norm activation. Argument r is the reason for the obligation (which has to include the norm id); g is the formula that represents the obligation (a state of the world that the agent must try to bring about, i.e. a goal it has to achieve); and d is the deadline to fulfill the obligation, expressed in terms of time.

It has to be remarked that the value expressed by the deadline d refers to the time regulated by a mechanisms inside the OE. This implies a notion of time related to the organization, which may be different with respect to the time used by agents and also with the time managed by other entities playing inside the application domain. As in other approaches, a norm is defined since a static / declarative aspect (when norms are declared in NPL) and a dynamic / operational aspect (when

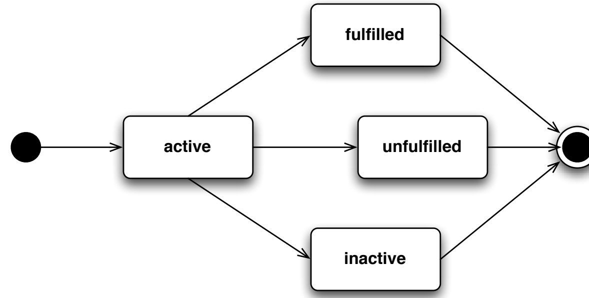

Figure 2.2: State transitions for NPL Obligations (taken from [61]).

an obligation is created, it is added to the organizational configuration). Thanks to the explicit notion of time inside the organization, it is possible to model a life-cycle of an instantiated obligation which can be monitored at runtime (Figure 2.2). In this view, the dynamic aspect of the OE results from the interpretation of NPL programs and the consequent creation of obligations for agents actually participating the organization[2].

Figure 2.2 informally describes the transitions involving obligation states (the interested reader can find a more detailed description of the dynamic aspects of the resulting normative system, including operational semantics, in [61]). An obligation is instantiated when the activation condition φ of some norm n holds. The activation formula is then used to instantiate the variables a, r, g and d for the obligation to be created. Once created, the initial state of an obligation is *active*. The obligation state changes to *fulfilled* when agent a fulfills the norm obligation g before the deadline d. The obligation state changes to *unfulfilled* when agent a does not fulfill the norm obligation g before the deadline d. As soon as the activation condition of the norm that creates the obligation φ ceases to hold, the state changes to *inactive*.

Notice that a reference to the norm that led to the creation of the obligation is kept as part of the obligation configuration itself (the r argument, that includes the norm *id*). This reference is kept in order to refer the norm instance at runtime. Besides, the activation condition of this norm must remain true for the obligation to stay active On these basis, only an active obligation will become either fulfilled or unfulfilled, when the deadline comes.

[2]The model will be detailed in Chapter 7, where a programming example will be discussed.

Norms management

As emphasized by many authors (among others, [56]), the mechanisms for application of norms are an integral part of norms specification. In the context of the OE described here, norms are implemented by means of mechanisms that instrument them in the interaction space, thereby in the context where norms have to be considered by agents participating the organization. The use of a norm based OE envisages two main mechanisms promoting the application of norms [71]: regimentation and enforcement.

As introduced in Subsection 2.2.5, *regimentation* is a mechanism that simply prevents the agents to perform actions that are specified as forbidden by a norm. When action regimentation is of concern, it seems a reasonable idea to make the OE to manage such a regimentation. Therefore, actions which typically refer to the OE – as "commit to mission" or "role adoption", for instance – can suitably regimented by the OE itself.

On the other hands, *enforcement* mechanisms assume the fulfillment / unfulfillment of the norms to be detected, evaluated as a possible violation, and then judged as worth of sanction / reward or not. In this case the possibility for the OE to apply enforcement is limited by the need of activities which are further required after the norm violation. Although the OE includes the mechanism allowing the detection of a given violation, the organizational entity needs the execution of additional activities (i.e. judgement, sanctioning) which are not in charge of an infrastructure. It may be deemed as necessary, in this case, the support of additional agents inside the organization to fulfill enforcement.

Norms application

As illustrated in Figure 2.3, two kinds of mechanisms for applying norms can be distinguished: management in terms of checking norms compliance before finalizing changes in the organizational states; management in terms of detection of fulfilments, statement on possible violation and possible sanctioning (or rewarding) activities. As in the former case the mechanism refers to regimentation strategies[3], in the latter case the mechanisms refers to enforcement.

An important aspect to remark in norms management is the role played by organizational entities once enforcement is of concern. While detection can be implemented as an automatic process, namely the application of a procedure that

[3]Different approaches implementing regimentation inside the OE are described in the next chapters.

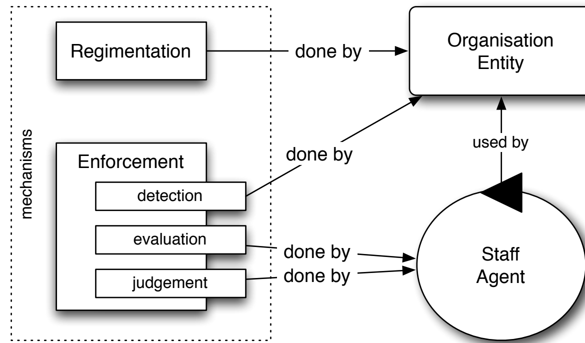

Figure 2.3: Norms management inside an organization is automatically ruled by the OE in case of *regimentation* and by organizational agents in case of *enforcement* and judgement about violations (modified from [63]).

does not require decisions, the evaluation and the judgement after some violation could require deliberation and reasoning. As said, whereas the functionalities provided by the OE allows to detect and show the violation of a norm, evaluation and judgement are tasks more suitably implemented by agents inside the organization. In this view, the OE allows agents to be informed of current status of the norms in order to evaluate the existence of possible violations and then decide regarding the related measures to take. To stress the embedding of dedicated reasoning related to the management of the organization these agents have been called *organizational agents* [63]. In general, they can be assumed as agents aware about the organizational schemes – or as agents embedding specific organizational knowledge. Their missions are essentially related to monitoring and controlling the norms inside the organization. Such activities typically include observing OE states (institutional states) and possibly intervening on the need, by changing and adapting the infrastructure or interacting directly with other agents to be sanctioned or rewarded (see Figure 2.3). As noticed in Subsection 2.2.5, such an organizational role has already been proposed in many OMIs [15, 2, 18]. By adopting the $\mathcal{M}$oise model, one or multiple schemes can be introduced for defining the activities of organizational agents in terms of missions, which are in turns responsible for monitoring of violations. Once the norms regulating the various roles are violated by agents, organizational agents detect the violation and then apply enforcement through rewards or sanctions in order to promote as much as possible the norm compliance.

2.4 Final Remarks on Organizations in MAS

The perspective showed in this chapter emphasizes notions, concepts and the mechanisms already adopted in conceiving organizational entities in MAS. As seen, most of the approaches are established along notions which are forged on the metaphor of human organizations. These notions includes, for instance, roles, groups, norms, shared plans, collective intentions and so on. The concrete realization of such a modeling approaches shift these abstract notions into programming models, providing a series of language primitives and constructs allowing flexible development and management of organizations inside a MAS.

In the second part of the chapter the $\mathcal{M}$oise OML has been described. It allows to express an organizational specification taking into account multiple dimensions, thereby introducing a meta-level aimed at coordinating the activities of heterogeneous agents in complex organizational patterns. The concrete organizational entities realizing $\mathcal{M}$oise organization inside a MAS make use of an additional component, by which the specification originally placed in multiple conceptual dimension can be translated to a normative language. The aim of the translated specification is to take all the organizational constraints into account in managing the organizational configuration at runtime. A further relevant characteristic of $\mathcal{M}$oise organizations is that it allows two kinds of agents to evolve inside the system. We referred to them as participant agents, which are agents using the organizational functionalities in order to coordinate each other and fulfill shared goals, and the organizational agents, which are agents aware of organizational structures and are typically aimed at detecting norms violations and possibly at promoting a desired organizational evolution.

Besides the realization of coherent organizational entities, recent trends in MAS are addressed at reconciling organizations with the concrete work environment where agents are supposed to interact. These approaches accounts for better situating organizations inside the whole system, thus considering an additional dimension to be integrated aside the organizational one. In this view, the aim is to ground organizational entities within the specific conditions and context relieved inside the work environment, for instance in order to apply norms thanks to infrastructures supporting the organization and in order to promote automatic mechanisms controlling regimentation as well as enforcement strategies. The challenges related to these approaches, along with a structured approach to environment programming which is needed for their realization, will motivate the rest of this work and are resumed in the next chapter.

Chapter 3

Organizations Situated in MAS Environments

This chapter introduces environments as an explicit dimension among the ones to be envisioned in an organizational model. As working hypotheses, environments, to be considered as the set of external entities not modellable as agents but aimed at supporting their activities, accounts for a series of theoretical benefits, ranging from interactions, situatedness, monitoring of agents activity, strategies for regimentation/enforcement etc. On the other hands, considering environments for these purposes places a series of modeling issues, either on the computational approach to be adopted and on the programming approach to be followed to provide a seamless integration with other organizational functionalities. In this view, a series of important challenges are considered to provide a structured approach to organizations situated in computational environments. Finally, the Agents & Artifact meta model for MAS (A&A) is introduced as a valuable paradigm for the construction of environments as first class entity of agents worlds.

3.1 Situating Organizations in Computational Environments

MAS organizations are potentially a powerful tool to build complex systems where computational entities can autonomously pursue their activities exhibiting social attitudes, and being designed according to heterogeneous models, technologies and programming approaches. On the other hands, programming such complex systems may require agents to be situated in a well established computational

environment populated by different computational entities, not characterized by agentive properties as autonomy, pro-activity, social attitudes, etc. but already providing agents with supporting functionalities which agents can exploit to serve their purposes.

As seen in the previous chapter, the trend in building organizational systems inside a MAS is mainly addressed towards programming organizations as middleware entities not shaped according to the agent abstraction, but realized by software components (or objects) which agents can interact by using ad hoc primitives. Once such middleware based entities are deployed inside the MAS, an additional requirement is given by the need to "situate" the organizational entity within the workplace where agents are immersed. In general terms, this places the challenges to conceive the proper interaction patterns between the involved entities, namely: *(i)* how the organization should interact with other external environment entities (O-E relationship in Figure 1.1), *(ii)* how the organization, as an infrastructure inside MAS, should interact with agents sharing the same work environment (O-A relationship in Figure 1.1). Considering environments for these purposes places a series of modeling issues, either on the computational approach to be adopted and on the programming approach to be followed to provide a seamless integration with other organizational functionalities.

A second issue about situating organizations is whether to consider environments as an additional aspect (i.e. dimension) to be provided within the organizational specification, or, on the other hands, whether to consider the organization as a separate concern, to be integrated with an existing environment infrastructure. In the former option, the resulting organizational entity intrinsically includes both the environment and the organizational aspects, thus it can be conceived in as a coherent entity at design time. The latter options accounts environments as constituted by heterogenous entities which may exist independently from the organizational specification. Of course, once the organizational dimension is installed as an additional concern inside an environment, a series of functional relationship have to be established to define which kind of interaction may occur between the organizational entity and the environmental ones.

Independently from the concern attributed to the environment dimension in the context of the whole system, a series of important aspects have to be further considered in order to provide a structured approach to environments, including the type of data, the action and perception models, computational and distribution models, and so on. To these ends, the research lines that in recent years addressed the notion of environments as first class abstraction in MAS can be investigated to find a valuable contribution in building environments as places where an orga-

nization has to be straightforwardly situated.

The chapter is organized as follows: Section 3.2 surveys existing approaches dealing with organizations situated in computational environments, emphasizing strength and weakness of a series of research lines. Taking a programming perspective, Section 3.4 analyzes several aspects needed to provide a structured approach to environments. The final section of this chapter (Section 3.5) introduces the Agents & Artifact meta model for MAS (A&A) as a valuable paradigm for the construction of work environments structured by decentralised entities localized in distributed nodes and collected in coherent work places.

3.2 Environments and Organizations in MAS

Although early approaches in organization programming have not been addressed at identifying a specific role for computational environments, a recent trend in research has been dedicated at the challenge to situate organizations in concrete application domains. This section resumes works recently proposed in this direction, analyzing their strenght and weakness and pointing out a series of open issues and challenges.

3.2.1 Current Approaches

In order to reconcile physical reality with institutional dimensions, MASQ (Multi-Agent System based on Quadrants) introduced a meta-model promoting an analysis and design of the global MAS along different dimensions [132]. MASQ approach strongly relies on the AGR OML extended with an explicit support to environment as envisaged by the AGRE and AGREEN models [6]. The organizational approach takes into account both the environment and the institutional part of MAS societies, integrating different dimensions (agents, environment, interactions, organizations and institutions) into an integral view. In particular, four dimensions are introduced, ranging from endogenous aspects (related to agent's mental attitudes) to exogenous aspects (related to environments, society and cultures where agents are immersed). The same infrastructure used to introduce organizational entities is also regulated by precise rules for interactions between the involved entities. Those entities can be agents or objects, and are assumed to dwell the same work environment: the resulting interaction relies on Ferber's model of influences and reactions [51], which will be discussed later on this chapter as a general approach to conceive interactions inside a MAS.

One of the proposals promoting specific entities addressed at modeling organizations through their environment extension is due to Okuyama et al. [91]. In order to implement situated organizations instrumenting physical environments where social interactions are of concern, they proposed to distribute "normative objects" as reactive entities inspectable by agents working in "normative places". The notion of normative object is, in this case, related to the one of environment objects, thus conceptually close to the notion of object envisaged for instance in MASQ. As proposed by Okuyama et al., normative objects are computational objects with an additional informative content, which can be exploited by the organizational infrastructure itself to make available information about norms that regulates the behavior of agents within the place where such objects can be perceived by agents. On the other hands, agents operating in normative places have the chance to inspect normative objects so to learn the norms actually regulating their application domain. The approach is assumed to improve emergent dynamics governed by specific norms addressed at controlling agents behavior. In fact, normative objects are supposed to indicate obligations, prohibitions, rights and are indeed readable pieces of information that agents can get and bring about with no previous knowledge. Put together, normative objects and normative places are assumed to build a distributed normative infrastructure, allowing the definition of certain kinds of situated multi-agent organizations, in particular organizations that operate within concrete environments and in the context of simulative applications.

A remarkable aspect defined in the approach followed by Okuyama et al. is that situated rules can be locally defined to allow agents to implicitly interact with a normative institution, as for instance adopting roles in situated context. The mechanism is based on the fact that the reification of a particular state in a normative place may constitute the realization of a particular institutional fact (e.g., "being on a car driver seat makes an agent to play the role driver"). This basic idea is borrowed from John Searle's theories on speech acts and institutions. In his book "the construction of social reality" [128], Searle provides a theory on how social constructs like marriage or money can exist in a world consisting of brute facts regulated by natural laws as "physical particles in fields of force".

Aiming at an explanation of social phenomena, Searle distinguishes between *brute facts*, like the height of a mountain, and *institutional facts*, like the score of a baseball game. He argues that society can be explained in terms of institutional facts, and institutional facts arise out of collective agreements through special

kind of rules, that he refers as *constitutive rules* [1]. Constitutive rules constitute (and also regulate) an activity the existence of which is logically dependent on the rules themselves, thus forming a kind of tautology for what a constitutive rule also defines the notion that it regulates. Constitutive rules do not regulate already existing (brute/concrete) reality but are addressed to the definition of a new (social/institutional) reality. In the example of Okuyama, a sentence like "being on a car driver seat makes an agent to play the role driver" strongly situate the institutional dimension on the environmental one, it both regulates the concept of role adoption and, at the same time, it defines it.

Constitutive rules in the form *X counts as Y in C* are also at the basis of the work proposed by Dastani et al. to bridge the gap from environment states and normative states [33, 34]. In their approach, a normative infrastructure (which is referred as "normative artifact") is conceived as a centralized environment that is explicitly conceived as a container of *institutional facts*, namely facts related to the normative or institutional states, and *brute facts*, namely states which are related to the concrete, "physical" workplace where agents dwell. To shift facts from the brute dimension to the normative one the normative system is assumed to handle constitutive rules defined on the basis of "count-as" and "sanctioning" statements, which are specified in terms of transitions regulating the effects of the actions performed by the agents in their environment and allows the infrastructure to recast brute facts to institutional ones. In particular, the antecedent of each rules specifies the states needed for the rule to apply, while the consequent contains the effects defining how the rule is applied (for instance, "being on a train without the ticket count as violation" , "violations of being on a train without the ticket are sanctioned with fees"). The mechanism regulating the application of "count-as" and "sanctioning" rules is then based on a monitoring process established as an infrastructural functionality installed inside the normative system. Thanks to this mechanism, agents behavior can be automatically regulated through enforcing mechanisms, i.e. without the intervention of organizational agents.

A similar approach is proposed in the work by Tinnemeier et al. [135], where a formal model for a normative programming language based on conditional obligations and prohibitions is proposed. Thanks to the inclusion of the environment dimension in the normative system, this work explicitly grounds norms either on institutional states either on specific environmental states. In this case indeed the normative system is also in charge of monitoring the outcomes of agent activities

[1] The notion of constitutive rules, as opposed to regulative rules, has been introduced by Searle in studies on socio linguistics and speech acts [127].

as performed in the work environment, in so doing providing a twofold support either to the organizational dimension either to the environmental one.

The notion of observability of environment states is also at the basis of the approach to Situated Electronic Institutions (SEI), recently proposed as an extension of AMELIE. Besides providing a runtime management of the normative specification, SEIs are aimed at interceding between (real world) environments and the institution itself [18]. In this case, special governors, namely modelers, allow to bridge environmental structures to the EI by instrumenting environments with embodied devices controlled by the institutional apparatus. Participating agents can, in this case, perform individual actions and interactions (either non message based) while operating upon concrete devices inside the environment. Besides, SEI introduces the notion of staff agents, namely organization aware agents which role is to monitor ongoing activities performed by agents which are not under the direct control of the institution. Staff agents are then assumed react to norm violations, possibly ascribing sanctioning and enforcements to disobeying agents. Institutional control is also introduced by the mean of feedback mechanisms aimed at comparing observed properties with certain expected values. On the basis of possible not standard properties detected, an autonomic mechanism specifies how reconfigure the institution in order to re-establish equilibrium.

3.2.2 Open Issues and Challenges

Although some of the aspects enabling a conceptual integration between organizations and environments are already well established among the proposed approaches, as for instance the principle of constitutive rules, other aspects still seem far from being addressed. A series of open issues could be listed:

- There is no agreement in which kind of computational model to be adopted by environments. Instead, either a centralized approach (i.e. environment constituted by a unique entity, as in [33, 34, 135]) and decentralised approach (i.e. environment constituted by several independent entities as in [132, 18]) are adopted.

- Different approaches are provided for the interaction model between agents and environment. Besides, there is not a clear vision on how an environment should support agents in their native capabilities, as for instance the ones related to action and perception.

- The computational treatments of goals clashes different acceptations once they are referred to agents and their subjective goals, and when they are related to organizations and their global goals. By considering environments explicitly, either agents and organizations should be able to ground goals to actual environment configurations, thus recognizing the fulfillment of their objectives once the pursued goals have been reached in practice (this approach is adopted, for instance, in [33]). Other approaches, as for instance ORA4MAS [63], do not assume organizations able to automatically detect the fulfillment of global goals in terms of environment configurations.

- Similarly, a weak support is provided for grounding norms in concrete environment states, and for managing their lifecycle with respect to changing environments. No agreement is then established on which kind of monitoring and sanctioning mechanisms must be adopted. Some approaches envisage the role of organizational/staff agents [18], other approaches propose the sole automatic regulation provided by a programmable infrastructure [33, 34, 135].

- Few approaches accounts openness, for instance with respect to heterogenous agent architectures an towards protocols and type of data to be exchanged between the involved entities.

- A weak support is provided for distributing environments across multiple nodes and machines, where a distributed approach further require a more sophisticated management of invariants as time and global states.

- It is not clear which kind of capability, and which grade of awareness, is required for agents to exploit the functionalities provided by a (situated) organizational entity.

With the aim to make order among the above mentioned challenges, and in order to bridge the gap between the existing approaches, we argue that some of recent research on environment modeling and programming could be fruitfully explored. To this end, the next sections survey research lines that recently proposed the notion of environment as first class abstraction in MAS and suggest a principled approach to environment modeling/programming.

3.3 Environment as first class Abstraction in MAS

The notion of environment is more and more recognized as a pivotal concept in development of agent systems. Computational environments generally represent the virtual or physical place where agents are situated and where all the activities deemed for the agents to fulfill their design objectives take place. Fundamental features of the agent as autonomous computational entities can be directly or undirectly related to the environment: situatedness and reactiveness are, of course, obvious examples, but also pro-activeness, being the notion of goal, which is actually the essential aspect of pro-active behaviors, typically defined in terms of "desired" states of the world that an agent aims to bring about and realize in the environment concretely.

Actually two distinct perspectives can be adopted when defining the concept of environment in agent systems: a classical view, rooted in Artificial Intelligence [125], and a more recent one, grown in the context of Agent-Oriented Software Engineering (AOSE) [89]. In the classical AI view, the notion of environment is used to identify that external world (being the perspective centered on a single agent or on a set of agents considered as a whole) which is perceived and acted upon by the agents so as to fulfill their goals. A classical AI-oriented formalization of the notion of environment is provided by [148] where the notion of *task environment* is introduced. Task environment is described as a triple $Env = \langle E, e_0, \tau \rangle$ where E is a set of environment states, $e_0 \in E$ is an initial state, and τ is a state transformer function $E \times A \rightarrow E$, computing the new state of the environment given an agent action $a \in A$. This simple model is effective for studying agent strategies in doing tasks inside the environment, but too abstract and simple for the purpose of environment programming, since it does not directly capture aspects such as the concurrent work of multiple agents in the same environment, environment observability, environment processes.

These aspects are taken into the account in the seminal work by Ferber and Müller, who devised the notion of influences and reactions to model agent environment interactions [51]. In Ferber and Müller model, the overall system dynamics in decomposed in two parts, the dynamics of the environment and the dynamics of the agents situated in the environment. MAS evolution is described as the transformation of a *dynamical state*, defined as a tuple consisting of the state of the environment and the set of *influences* simultaneously produced in the environment. Influences come from inside the agents and are attempts to modify the course of events in the world. *Reactions*, which result in state changes, are produced by the environment by combining influences of all agents, given the lo-

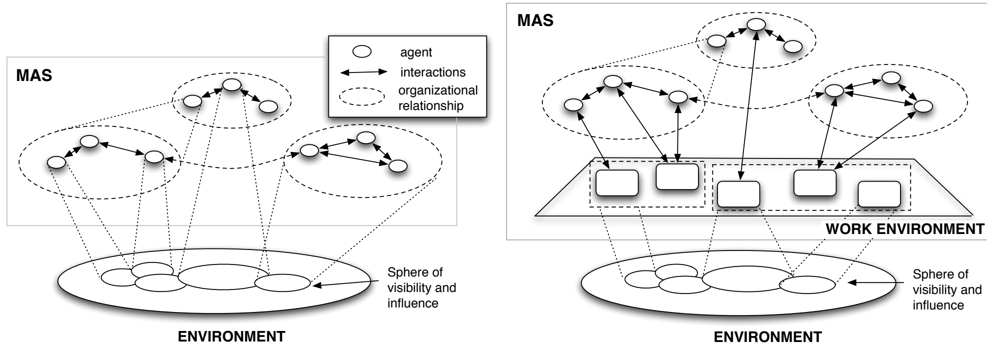

Figure 3.1: *(left)* A canonical view of MAS, adapted from [70]. *(right)* A MAS enriched with a work environment layer

.

cal state of the environment and the laws of the world. A *Cycle* function is used to formally define this evolution. A canonical representation of a Multi-Agent Systems (MAS) including the environment is shown in Figure 3.1 (left). The figure, as taken from [70], and it shows the environment as the context shared by agents in a MAS. Each agent has a *sphere of influence* on the environment, which is the portion of the environment that the agent is able to control—sphere of influences of different agents can overlap, meaning that parts of the environment can be jointly controlled by more than one single agent.

Besides this perspective, recent works in the context of AOSE introduced the idea of environment as a *first-class abstraction* for MAS engineering [144], a suitable place where to encapsulate functionalities and services to support agents activities (the interested reader can consult [145, 146] for a survey of the research works developed in this area). In this latter view, the environment is no more just the target of agent actions and the container and generator of agent percepts, but it represents an integral part of the system, that can be suitably shaped – or better, *designed* – so as to improve the overall effectiveness of the system. Thereby, the environment can be an effective place where to encapsulate functionalities and infrastructures that concern the management of agent interactions and their coordination. Environment as a first class abstraction is the locus to encapsulate external functionalities and services which are assumed to be exploited by agents possibly enabling (complex) interaction patterns. Those functionalities and services can be deployed to support agent individual and collective activities. This turns out to be a central aspect for defining and enacting into the environment

infrastructures improving interactions coordination, security, and, above all, organizations. Taking a design perspective, in [142] Weyns and Holvoet detail a complete formalization of a general abstract *architecture for situated MAS*. Their work sketches agent environment interactions in terms of influences and reactions – as introduced in [51] – and emphasizes the notion of locality, namely each computational agent is situated in his local context where it is able to perceive and in which it can act. The model includes the notion of *regional synchronization*, which makes it possible to avoid the need of a global synchronizer as used in Ferber and Müller (that results in a synchronous evolution of the MAS), and the support for *active perception* [147], which enables an agent to direct its perception at the most relevant aspects of the environment according to its current task, thus promoting situation awareness. The perspective adopted is intentionally addressed to the *architectural* aspects of the systems, leaving the programming issues on the background. The main elements of the abstract model concern the functional modules that are needed to provide the functionalities that are meant to be encapsulated in the environment: interaction, communication, synchronization and data-processing, observation and data-processing, translation [143].

It may be argued that both the approaches followed by Ferber / Müller and Weyns / Holvoet in some sense bias the viewpoint of the developer: indeed, the environment is seen as a single entity, whose design/programming means devising a unique functional model for the overall set of actions and percepts. Accordingly the definition of environment functioning is set by specializing its functional modules, according to the specific problem to solve (i.e., the system to build). The "monolithic" perspective is evident in the former formalization [51] introducing the global functions like *React* and *Exec*, and also in the latter model [142], which is based on global functions like $React$, $Apply$, $Compose$, $Collect$.

This perspective is reflected in existing agent programming languages and platforms, which typically provide a (weak) support to environments through constructs and API to define agent actions and perceptions and to enable the interaction with external systems. Frequently, such an API includes also a support for defining the structure and behavior for the environment – besides the interface allowing agent to exploit resources at runtime – so as to set up design-time simulations of the overall system prior to deployment. In this canonical view, which is rooted mainly in the Artificial Intelligence [125], the environment is conceived as a black-box, defining the set of allowed agent actions and the perceptions that can be generated.

As opposed to the centralized view, other approaches take a different perspective, based on the notion of work environments as dynamically composable set

of computationally independent entities aimed at structuring the environment in a decentralised and distributed fashion (Figure 3.1, right). This view of decentralised work environment in particular is the one proposed by the Agents and Artifacts (A&A) meta model, that is adopted as the reference model for environment programming in the context of this thesis. Before providing the details on the envisaged approach, the next sections analyze and emphasize a set of relevant aspects to be taken into account once programming environments, thus setting the stage for a structured approach to environment programming.

3.4 A Structured approach to Environments

As argued in the previous sections, in order to conceive work environments as interaction spaces which can be instrumented with opportune infrastructures aimed at promoting agent organizations, a structured approach to environment programming is required. As for structured approach, a series of important aspects have to be addressed. In this view, the next sections identify and discuss the key aspects that – we argue – characterize a structured approach to environment programming.

3.4.1 Action Model

Environment should support an action model, concerning aspects related to the mechanisms enabling agents to affect or change the state of the environment and accordingly receive feedback percepts. Action model accounts how agents repertoire of actions can be defined/programmed in order to enable fruitful Agent-Environment interactions (A-E relationship in Figure 1.1). Besides, the action model is also deemed to define which action execution model is adopted, thereby if either actions are modeled as *events*, i.e. as a single transition changing atomically the state of the environment, or they are *processes*, involving a start (triggering) event and an end (completion) event. Current agent programming languages typically model external actions as events, assumed to *atomically* read or change the state of the environment—as it is for instance in *Jason* and *2APL* [14, 31]. Other approaches are process-oriented, as in the case of the *influences and reactions* model introduced by Ferber and Müller in [51]—adopted in the MadKit agent platform [57].

Actually, the semantics adopted for action execution has a strong impact also on the synchronization functionalities that can be provided by the environment:

for instance, in order to define actions that allow for synchronizing agents with external entities, a process-oriented semantics has to be necessarily adopted.

3.4.2 Perception Model

From an agent perspective, environments should be viewed either as the mean enabling the execution of external actions, either as the source of multiple perceptions, which filtering allows to control their ongoing activities. In this view, environments have to support agent perceptive activities, thus define mechanisms general enough for enabling perception to a wide range of signals to agents. This aspects relates on how environment should be perceived by agents, including the unambiguous definition of the stimuli generated by the environment and the corresponding agent percepts computed as result of the perception process.

Essentially, two basic semantics can be adopted in defining the perception model, *state-driven* and *event-driven*. In the former, stimuli are information about the actual state of the environment, and are generated when the agent is engaging the perception stage of its execution cycle[2]. In the latter, stimuli are information about changes occurring inside the environment, and are generated when changes occur, independently from agents' execution state. Typically, the perception process is executed internally by agents, so to compute the set of percepts given the set of stimuli coming from the environment. This places the problem to support, at an environment level, a wide spectrum of agent models, which can be shaped on different mechanisms supporting external stimuli. For instance, the model adopted to define agent abstract architecture [148], where a *see* function is introduced to model the perception process mapping an environment state E into a set of perception I, is state-driven. Referring to concrete agent programming languages and their formal operational semantics, *2APL* [31] and *Jadex* [112] adopt an event-driven semantic, while *Jason* a state-driven one [14]. Actually the chosen semantics have a strong impact on the dynamics of MAS program execution. For instance, the possibility for agents to control external actions during the whole execution is almost neglected in current agent programming models. Otherwise, in a state-driven approach, if the environment changes multiple times between two subsequent occurrences of the perception stage of an agent execution cycle, such changes are not perceived by the agent.

[2]Here we refer to agents processes as conceived in terms of execution cycles. In particular we refer to the abstract agent architecture as defined in [148] and the related agent processes.

3.4.3 Computational Model

Environment programming accounts for conceiving a computational model defining a set of environment states and functionalities. In general terms, environment may include two kinds of computations, namely those processes that are directly elicited by agent actions, and those that represent inner (automatic) processes inside the environment.

A main issue in defining environments computationally is in preserving as much as possible the abstraction level adopted by agents. This accounts that the concepts used to define environments, as well as the constructs used to program their structures and dynamics, should be consistent with agent concepts and their semantics. Working at balanced level of abstraction means, for instance, conceiving interactions based on primitives which are native for agents models, following similar approach to the ones already adopted as in the case of message based communication and ACLs. On the other hands, working a not balanced level of abstraction, means enabling agents to interact at a mechanism level, for instance constraining the use of object oriented components, where the interaction is based on method invocations and the agentive notions of action and perception simply make no sense.

A second aspect concerns the *concurrency model* adopted, that is how many threads or control flows are exploited to execute environment computational processes, and, in the case of multiple threads, what is the mapping with respect to the environment computations and how concurrency problems, such as race conditions and interferences, can be avoided. Clearly, this aspect strongly impacts on the performance of the whole system.

A main issue in environment computational model is related to the composition approach to the overall computational state/behaviour of the environment. Two main view have been envisaged so far dealing with environment composition, which we refer as centralized and decentralised approaches.

Centralized Approaches

The centralized approach, in which the computational structure and behavior of the environment is represented by a single, monolithic, computational object, with a single container of environmental states. The a centralized object approach envisage the environment as the entry point for defining the effect of actions and the set of stimuli generated. As seen in the previous section, this perspective is adopted by Weyns and Holvoet in [143], where a reference model for an abstract

– even centralized – environment architecture is proposed. It consists of a set of *modules* that represent core functionalities of a centralized environment component, and it provides the specification of the functional relationships between these modules. The decomposition is primarily driven by the way agents interact with the environment. An agent can sense the environment to obtain a percept, an agent can perform an action in the environment (i.e., attempting to modify the state of affairs in the environment), and it can exchange messages with other agents.

Agent programming languages as *2APL* and *Jason* natively adopt the same centralized approach, by providing a API to program an environment as an external component typically realized according to the mechanism of a Java class.

Decentralized Approaches

A different approach accounts for explicitly defining first-class structures (and finally abstractions) to decompose and decentralize the functionalities of the environment. A main example in this case is provided in the context of Distributed Artificial Intelligence (DAI) by the Lifeworld model introduced by Agre and Horswill in [1]. In contrast to traditional AI models of environment – such as task environments – Agre and Horswill recognize the importance of properly modeling agent-environment *interactions*, including also the *conventions* and *invariants* maintained in the environment by agents throughout their activity. In this view, a lifeworld is a description of an environment in terms of the customary ways of structuring the activities that take place within it – the conventional uses of tools and materials, the "loop invariants" that are maintained by conventional activities, and so on. As a main aspect of Lifeworld computational model, the notion of *artifacts* is introduced, as referred to the (non agentive) tools that evolved inside the system according to a "fitness" related to the support they can provide to agents. Lifeworld artifacts are thus arranged in the environment in ways that simplify works and reduce the cognitive burden on individuals for fulfilling their tasks.

A similar notion of artifacts is devised by the A&A model [99], which proposes a decentralized model of MAS environments based on the notion of artifacts and workspaces. The A&A approach will be described in next section and is at the basis of the approach proposed in this thesis. Other examples of decentralised environment include a generic notion of *object* as abstraction to conceive environmental entities. Once the notion of environmental object is of concern, the action model may include the actions to create, dispose and replace environment objects at runtime. Environment objects are adopted, among others, in GOLEM [17]

and MadKit [57]. MadKit has been one of the first general-purpose Java-based frameworks for developing agent systems, implementing either AGR organizations and the influence and reaction model devised by Ferber and Müller in [51]. Although not explicitly introducing a computational and programming model for the environment, MadKit allows for programming environmental *objects* with an associated computational behavior. Actually this programming support has been exploited in particular for defining the behavior of the environment in multi agent social simulations – including organizational patterns – implemented on top of MadKit [132]. Recently proposed as logic-based framework for programming MAS, GOLEM allows for representing an agent environment declaratively, as a composite structure that evolves over time. As in AGR and A&A, GOLEM envisages two main main classes of entities dwelling environments, namely agents and *objects*, which can be localized in *containers*. Besides being described in a logic-based framework, the features of the objects and containers strongly resemble those of artifacts and workspaces as placed in A&A. Interactions between these entities inside a container are specified in term of events whose occurrence is governed by a set of physical laws specifying the possible evolutions of the agent environment, including how these evolutions are perceived by agents and affect objects and other agents in the environment.

3.4.4 Internal Dynamics

Unlike agents, entities dwelling environments are passive, thus neither proactive nor autonomous. Their evolution is entirely determined by the environment specification and by the events that occur in it. Besides, environment entities can exhibit an automatic functioning for instance to specify activities in a timely fashion as clocks, temporal mechanisms, guards, etc. In this view an environment entity can have internal processes, which can be expressed from time to time by a dynamic state described by an environment program and a set of evolution laws like the transformer function in [148].

It has to be considered that the various dynamics taking place inside the environment can interfere each other, thus exhibiting functional links between the involved entities. Moreover, an environment dynamics can be governed by more general laws determining the global rules of a given environment. For instance, in the context of a pheromone infrastructure, such global laws can determine evaporation, diffusion and aggregation phenomena [102], or in a simulation of a physics environment such global laws can specify what happens once two objects collide [58].

3.4.5 Data Model (and Openness)

An environment programming model should be as much orthogonal as possible with respect to the models, architectures, languages adopted for agent programming, so as to naturally support system openness. Emphasizing orthogonality means *separation of concerns*: On the one side, agents are the basic abstraction to design and program the autonomous parts of the software system, i.e. those parts that are designed to fulfill some goal/task[3] – either individual or collective – encapsulating the logic and the control of their action. On the other side, the environment can be used to design and program those parts of the system which are *functional* to agents' work, namely those parts that agents can dynamically *access* and *use* to exploit some kind of functionality, and possibly *adapt* to better suite to their actual needs.

An important aspect related to orthogonality concerns the types of the data exchanged between agents and environment structures, which is used in particular to encode action parameters, action feedbacks, the informational content of stimuli (percepts), and their representation. To deal with environment data model, a mechanism for *data-binding* must be specified, defining how the informational items defined in the environment can be translated into the specific data model adopted by the agent (and vice-versa).

A further issue that must be faced in the case of open system is the definition of data models that allows for describing proper *ontologies* so as to explicitly define the semantics of the data involved in agent-environment interaction. To this purpose existing work in the context of Semantic Web and the models/languages adopted for describing ontologies (such as OWL) can be suitably exploited.

3.4.6 Distribution Model (and Localities)

The distribution model concerns how to program environments that need to be distributed (or can opportunistically be distributed) among multiple network nodes.

To this end, a distributed environment model may introduce an explicit notion of *workplace* (i.e., locality) to define a non-distributed portion of the computational environment. Accordingly, it has to be defined if and how different workplaces – and related computational structures – are connected and may eventually interact each other. On the agent side, the adopted distribution model affects the repertoire of available actions, possibly including also actions to enter into a place or move from place to place.

[3] Here the concept of task and goal are used as synonyms.

Actually, the environment distribution model affects also the *time model* which can adopted inside the MAS. It can be showed that for distributed MAS it is not feasible – both from a theoretical and practical point of view – to have a *single* notion of time inside the system. In other terms, is it not feasible to mark with time-stamps multiple events across distributed workplaces and then define a global notion of order among such events[4]. This is a main issue, since many formalizations of agent systems in different contexts – such as e-Institutions, normative systems, agent organizations – typically are based on a global notion of time. As explained in the following of this work, the approach followed to deal with a coherent notion of time is in subdividing an environment into not distributed sub-environments defining a spatial-temporal localities, where it is possible to recover an ordered notion of global time at the level of the single place.

3.5 Agents & Artifacts

The approach to MAS design and programming followed in this thesis is grounded on the A&A (Agents and Artifacts) meta-model, which adopts artifacts and workspaces – along with agents – as the basic building blocks to program MAS and, more generally, to engineer complex and distributed software systems [99]. Before providing in Chapter 4 a detailed description of CArtAgO as the computational platform allowing to build MAS in A&A terms, this section briefly resumes some of the underpinnings of A&A as a (meta) modeling approach. After having briefly resumed the theoretical foundations in Subsection 3.5.1, Subsection 3.5.2 introduces the concepts at the basis of A&A.

3.5.1 Foundations

The role of artifacts in the context of human activities – social activities in particular – has been deeply investigated and crossed the boundaries of several disciplines as Activity Theory (AT) [80] and Distributed Cognition [77], already having a counterpart in heterogenous fields of computer science, like CSCW and HCI [133, 85].

According to AT, any activity carried on by one or more collaborating individuals cannot be conceived neither understood without considering the tools (or artifacts) that enable interactions and mediate the actions between the involved

[4] This may be due to network delay or concurrent processes which are concurrently in execution on different nodes.

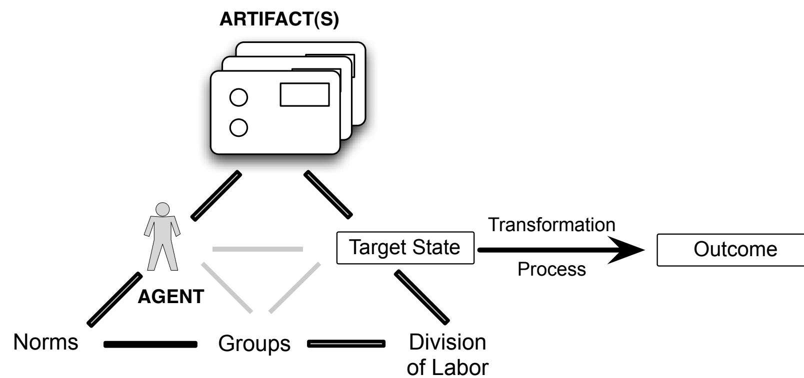

Figure 3.2: A schematic representation of human activities in relation with the role played by artifacts, norms and communities, as placed by Activity Theory.

entities (Figure 3.2). Artifacts mediate the interaction between individuals and their environment (including the other individuals); they reify the portion of the environment that can be designed and controlled to enable and support either individual and collective activities. Moreover, as an observable part of the environment, artifacts can be monitored and controlled to assess the overall system performance. Using artifacts modifies their states, thus keeping track of artifacts usages accounts for tracing the system history. In other words, mediating artifacts become first-class entities for both the analysis and synthesis of individual as well as cooperative working activities inside a complex system. Such a vision is also promoted by Distributed Cognition [77], a branch of cognitive science claiming that human cognition and knowledge representation, rather than being confined within the boundaries of an individual, are distributed across individuals, tools and artifacts in the environment.

The same degree of complexity envisaged within social systems accounted by AT and Distributed Cognition can be found also in MAS. This is why we can consider those inter-disciplinary studies as fundamental ones for the analysis and synthesis of social organizational activities inside agent systems, in particular once an explicit use of artifacts is assumed to mediate such activities [115].

3.5.2 Meta-Model for engineering MAS

By drawing inspiration from Activity Theory [85], the notion of *artifact* in MAS has been introduced the first time in [100, 115] in the context of MAS coordina-

tion, generalizing the notion of coordination media in particular to define the basic properties enabling and managing agent interactions. The concept has been then generalized besides the coordination domain, leading to the definition of the A&A (Agents and Artifacts) meta-model [99, 121].

Using the terminology introduced by Daniel Dennett [35], A&A takes a *design stance* in modeling MAS environments and is actually a conceptual model—a so-called *meta-model* in software engineering terms. On the basis of the A&A model, the CArtAgO platform [122, 119] has been proposed as the development framework supporting programming and execution of environments built upon the notion of artifacts[5]. A&A perspective emphasizes the role of work environments. Built in terms of artifacts and localized in terms of workspaces, a work environment can be perceived and used by agents as first-class entity of their world, and is explicitly designed and programmed by MAS designers so as to ease agent activities – in particular those involving interaction, coordination, and cooperation. Hence, work environments can be represented as an extra computational layer in the MAS, between agents and the external environment, mediating agent activities but also agent interaction with the external environment (see Figure 3.1, right)[6].

In the A&A meta-model, agents are the basic abstractions to represent active, task-/goal-oriented entities, designed to pro-actively carry on one or more activities toward the achievement of some kind of objective. In order to fulfill their goals, different levels of capabilities and reasoning models can be adopted by agents.

On the other hand, artifacts are the basic constitutive structures of environments. Artifacts are modeled as abstractions to represent passive, function-oriented entities, and are assumed by the MAS designer to be exploited by agents. Taking the human society as a metaphor, agents play the role of humans, while artifacts coincide with the objects and tools (called artifacts in the human society, too) used by humans as either the means to support their work and achieve their goals, or the target of their activities. Artifacts can be used by agents to attain particular outcomes, either individually or cooperatively, during their working activities. Besides, artifacts can be exploited by agents to externalize part of their activities. Accordingly, artifacts can be observed by agents, i.e., with the aim to obtain

---

[5] A detailed description of CArtAgO and its computational model is given in Chapter 4.

[6] The notion of work environment has been recently refined in the one of *endogenous environment* to emphasize the pivotal role assumed for supporting agent activities in terms of artifacts and workspaces [120]. The notion of endogenous environment is opposed to the one of *exogenous environment*, which is assumed to collect those external entities which are not directly accessible by agents.

some relevant data and, even more, artifacts can be created and modified by agents according to the needs.

As a third abstraction introduced by A&A, workspaces represent a notion of not distributed locality inside the work environment, grouping together coherent set of agents and artifacts inside a specific application domain.

As a principle of modularization, A&A introduces *artifacts* as basic building blocks to shape *decentralised* work environments, representing *resources* and *tools* that agents can exploit inside workspaces to support their individual and social activities [99, 121]. As detailed in the next chapters, the artifact programming model defines the basic set of features and mechanisms that can be used on the one side by MAS programmers to shape the environment of the MAS, and on the other side by agents, to dynamically construct, use and adapt their environment. So the environment is not conceived as a single block, but as a dynamic composition of artifacts organized in workspaces localities: the resulting system assumes the appearance of a decentralized ecosystem, which evolution is characterized by societies of agents at work with multiple artifacts.

Properties

Unlike agents, artifacts are not meant to be autonomous or exhibit a pro-active behavior, neither to have social capabilities. Among the main properties that are useful according to artifact purpose and nature [97], one could list:

Inspectability and *controllability* are related to the agent capability of observing and controlling artifact structure, state and functioning at runtime, and of supporting their on-line management, in terms of diagnosing, debugging, testing;

Malleability (or *forgeability*) is the artifact property to be changed / adapted at runtime (on-the-fly) according to new requirements or unpredictable events occurring in the environment;

Linkability is the capability (owned by agents) of functionally link together distinct artifacts at runtime as a form of dynamic composition, as a means to scale up with complexity of the function to provide, and also to support dynamic reuse.

Situatedness is, the property of being immersed in the MAS work environment, and to be reactive to environment events and changes.

Most of these features are not agent features: typically, agents are not inspectable, do not provide means for malleability, do not provide operations for their change, and do not compose with each other through operational links. On the other hand, agents are typically told to be situated: however, how this is realized, in particularly how pro-activity and re-activity features could be reconciled, is not an easy matter. It is worth to remark that, once artifacts are situated, agent situatedness could be recasted in terms of their interaction with artifacts.

The global picture of A&A is given by agents distributed across the network that inter-operate and coordinate both by communicating via some kind of ACL (agent communication language, i.e. FIPA ACL [52]) and by sharing and (co-)using infrastructures based on decentralised artifacts. Generally speaking, the A&A meta-model *recasts the space of interaction* within MAS, such that the entities of a system are envisaged to interact each other in three different ways: *(i)* agents *communicate* with agents through ACL; *(ii)* agents *use* and/or *observe* artifacts; *(iii)* artifacts *link* with artifacts. Adopting environment as an explicit design and programming dimensions makes it possible to rethink the strategies adopted to solve problems and build agent systems, both from the viewpoint of the individual agent and of societies of agents. A remarkable example concerns agent coordination: besides using strategies based solely on message passing and ACL-based communication protocols, in some cases it can be useful and more effective to introduce suitably programmed *coordination artifacts* [100], such as blackboards, shared maps or synchronizers – straightforwardly implementing coordination solutions based on mediated interaction. In this context, a main research investigation concerns the integration of CArtAgO with rational (intelligent) agents architectures and languages – the BDI model [114] being a main example – and the cognitive use of artifacts, towards scenarios where agents autonomously decide at runtime what to do based on which available artifacts to use/observe.

Coordination Media

The notion of mediated interaction and the introduction of proper *coordination media* as first-class abstractions to design the agent interaction space is a cornerstone of the research on coordination models and languages [28, 55]. The *tuple space* model and the related LINDA coordination language [54] are main examples, exploited today by industrial technologies for the development of distributed systems. These research works strongly influenced A&A and CArtAgO, besides Activity Theory. Thereby, the artifact abstraction was inspired by *programmable coordination media*, like tuple centers [93] adopted in the TuCSoN coordination

infrastructure [101]. Tuple centers have been used as concrete model to implement the concept of *coordination artifact* [98], which has been generalized then into the notion of artifact by CArtAgO, with the introduction of specific computational and programming models uncoupled from tuple centers. In human societies, coordination artifacts are as common as traffic lights, street signs, post-its on white-boards; in computational systems, things like blackboards, event-services, shared message boxes, could be easily seen as coordination artifacts.

In the context of MAS and A&A, coordination artifacts are used to both *enable* and *govern* forms of *mediated interaction* (i.e., where agents do not communicate directly but through a medium). Mediated interactions is essential to support forms of communication that are uncoupled along the time and space dimensions. Examples range from coordination abstractions such as tuple centers [94], to pheromone infrastructures [102] in the context of stigmergy coordination, to the governor entities in electronic-institution approaches [48, 47], to cite some. Compared to the basic notion of (programmable) coordination medium, the artifact abstraction can be considered a *generalization* beyond the coordination purposes, introducing to this end new key concepts such as the usage interface, a notion of observable state and inspectable meta descriptors as manuals; on the other hand, an artifact can be considered a *specialization* of the concept of coordination medium in the context of MAS and agent-oriented programming, with specific features that are deemed to be exploited by a strong notion of agents as defined in existing agent programming languages and architectures (i.e., not simply by processes, or actors, of a distributed system).

A similar notion pushing agent oriented coordination has been recently proposed by Campos et al., suggesting a layered infrastructure in order to promote coordination at different conceptual levels [19]. This approach introduces the notion of *Coordination Support*, according to which organizational entities can be placed aside other coordination services, each abstracting a particular set of functionalities aimed at easing agent development and management. In this view, coordination layers are conceived as an integrated infrastructure including either mechanisms *enabling* interactions, either functionalities aimed at promoting and *assisting* cooperation and organizational patterns among agents. In the former layers, the infrastructures are assumed to provide transport and connectivity functionalities, as well as infrastructures enabling communication between agents. The latter layers are assumed to provide organizational services modeled on different conceptual dimensions and assistance services, promoting improved of interactions in terms of agent adaptiveness and learning.

Decentralised Approach to Environments

Artifact-based environments introduce a decentralised modularization with respect to the one promoted by the reference architecture introduced by Weyns and Holvoet. In their reference architecture [142], modules represent basic blocks installed inside the software architecture of the environment. In this view, modules encapsulate some functions that developers can reuse and specialize when developing concrete environments for specific applications. But at an application level, and from the agent view point, the environment is still a monolithic entity, providing actions to act upon it and producing stimuli. In artifact-based environments, artifacts are modules both from the MAS developer viewpoint and from the agent viewpoint, while each single artifact is assumed to manage most of aspects dealt with by all modules of the reference model.

In A&A artifacts encapsulate some kind of *function*, however not from the MAS engineer point of view as in Weyns and Holvoet work, but from the agent point of view. For this reason, the agent-environment interaction model in artifact-based environment can be refined beyond actions and percepts as in the reference model. As it will be clarified in the next chapter, this can be done by introducing a basic set of actions by which agents can manage artifacts as an external resource. The actions allow agents to dynamically use, observe, instantiate and dispose the modules (the artifacts) In the model by Weyns and Holvoet and in related specific models implementing it, typically modules are static, even if customized according to the specific application needs.

3.6 Final Remarks on Situated Organizations

This chapter concludes the survey on MAS approaches to organizations by emphasizing the role that computational environments can play to improve the functionalities of an organizational entity. Tackling the environment dimensions explicitly promotes situated kinds of organization, at the same time augmenting the capabilities for agents to interact with the organization, and the alternative strategies that an organization has to monitor and control the activities performed by agents inside the system. Whether the aim is to provide a seamless integration between the programming models adopted by possibly heterogenous agents and organizations specified along multiple dimensions, the environments play a pivotal role. To this aim, a structured approach to environments has been envisaged as a main challenge, and it has to be taken into account in order to face with the multifaceted

structures and dynamics that a complex systems concern.

As recent contributes to MAS research emphasized, considering decentralised environments as a coherent entity concurs in establishing a series of theoretical advantages, which can be found either by the developer either by the agent finally exploiting the system. In this view, the A&A modeling approach has been envisaged as an applicable approach to design environments in MAS, based on the notions of multiple artifacts and distributed workspaces.

The next two parts of this thesis focus on specific aspects of environments and organizations respectively, and provide a concrete programming approach aimed at conceiving dedicated infrastructures to be instrumented inside the system, both on the environmental and the organizational perspectives. Together with Moise, A&A is the approach that will be used as reference model, while the route to a unifying approach reconciling organizations, agents and environment, will be faced from a programming perspective.

Part II

Developing Environment Infrastructures based on Artifacts

Chapter 4

Environment Programming in CArtAgO

This part of the thesis describes a computational model and a related technology for MAS environment programming based on the A&A design approach. Both the design and the programming models envisage agents *as the basic abstraction which has in charge the autonomous part of the systems,* artifacts *as the abstraction which has in charge the functional part of the system, and* workspaces *as the abstraction providing a concrete locus for defining application domain, grouping together coherent set of agents and artifacts.*

In this chapter, CArtAgO is described as a concrete framework providing programming constructs, and run time support, for building distributed work environments based on agents, artifacts and workspaces. Interaction involving agents and artifacts are analyzed taking different perspectives and through the discussion of concrete examples. Basic interaction are enabled since the definition of a basic set of actions extending agents' repertoire with the capabilities needed to operate within CArtAgO. According to the reasoning capabilities owned by agents, the interactions may be conceived also in a more complex fashion, thereby enabling the so called "cognitive" level.

4.1 Taking the Environment Programming Perspective

The approach followed in this work emphasize the notion of *environment* as a first-class abstraction in agents and multi-agent systems. Environment is viewed

as the computational or physical places where agents are situated, providing the basic ground for defining the notions of agent perception and action, also reifying the global interaction space of any application domain.

As resumed in Section 3.4, recent approaches to MAS engineering pointed out the notion of environment as an effective place where it is possible to combine effective infrastructures concerning either services and functionalities exploitable by agents and aimed at easing their tasks, either those components aimed at supporting the management of agent interactions and at promoting agent coordination. This chapter shift the perspective from MAS design to *MAS programming*, emphasizing the notion of *environment programming*. The followed approach deems environment as a first-class abstraction not only at design time, but also in programming multi-agent systems. In this view, the environment becomes a pivotal element of the whole system either either at a design and at an implementation phase of development. The "Agents and Artifacts" (A&A) conceptual model puts forward the notion of *work environment*, seen as a computational environment that is part of the MAS, and which is designed so as to be a proper and effective place for agents to live and work [99, 121]. A&A is actually a so-called *meta-model* – representing design orientations and methodologies – to be mainly exploited for MAS design. CArtAgO is its accompanying technology, providing both a concrete computational model and programming platforms aimed to fully exploit the benefits of the A&A approach up to MAS programming and development. To this end, in this chapter we describe CArtAgO, including the most recent features, as a platform/infrastructure that introduces: *(i)* a concrete computational model to define artifact structure, computational behavior, and agent-artifact interaction, *(ii)* a programming model to concretely code artifact-based environments and to allow agents to work in them, and *(iii)* a runtime environment to support artifact-based environment execution.

Accordingly, a series of research challenges are envisaged: *(i)* the definition of general-purpose computational and programming models and related technologies (languages, frameworks) to program the environments and their entities, and *(ii)* their integration with existing agent programming models and frameworks. The approach allows for specifying an environment in terms of a dynamic set of first-class computational entities called *artifacts*, collected in localities called *workspaces*. Artifacts represent resources and tools that agents can dynamically instantiate, share and use to support their individual and collective activities [99, 121]. Artifacts are first-class abstractions for both designers and programmers of MAS, which define the types of artifacts that can be instantiated in a specific workspace, defining their structure and computational behavior. On the other hand, artifacts

are first-class entities of agents world, which agents may perceive, use, compose, and manipulate on the need to serve their purposes.

The chapter is mainly based on recent works, as they are cited across the various sections (among others [122, 121, 117, 109, 119, 118]). Those contributes have been revised and extended so as to include the most recent improvements, in the environment programming perspectives and challenges.

In the first part of this chapter, an abstract overview of the basic concepts underlying the artifact abstraction and the basic actions required for agents to interact with artifacts is provided in Section 4.2, along with the a description of the capabilities needed by agents to work in artifact based workspaces. Then, the CArtAgO technology that makes available a concrete programming framework to build artifact based work environment is introduced in Section 4.3. The second part of this chapter shows examples of programming MAS in CArtAgO, in particular using *Jason* ([14]) as the reference model and language for specifying agents (Section 4.4). The notion of cognitive interaction is introduced in the last part of the chapter, by describing advanced aspects in programming agent artifact interactions (Section 4.5). For instance, agents may decide to exploit artifacts in order to improve their epistemic states, or to use artifact to externalize part of their goal oriented activities. Besides, the functionalities provided by artifacts can be internalized, this enabling agents to exploit artifacts which are unknown at design time. Finally, Section 4.6 provide final remarks, situating the contribute of this chapter in the context of the overall work.

4.2 Artifact-Based Environments

Placed in A&A terms, a MAS environment is conceived as a dynamic set of computational entities called *artifacts*, representing special purpose resources and tools that agents can share and exploit to serve both their individual and collective purposes. The overall set of artifacts can be organized in one or multiple *workspaces*, possibly distributed in different network nodes. A workspace represents a work place inside the MAS, namely the locus of one or multiple activities involving a set of agents and artifacts which may be addressed at reifying an application domain. Figure 4.1 resumes an overview of the main concepts characterizing artifact-based environments.

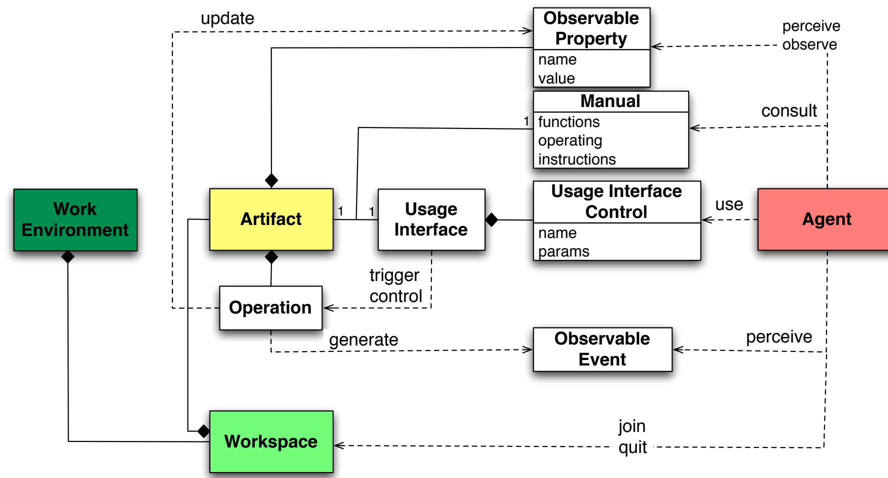

Figure 4.1: Entities involved in the A&A meta-model of MAS, here expressed in UML-like notation.

4.2.1 Artifact Computational Model

From the MAS designer and programmer viewpoint, the notion of artifact is a *first-class abstraction*, the basic *module* to structure and organize the environment, providing a general-purpose programming and computational model to shape the kind of functionalities available to agents. Actually, MAS programmers define *types* of artifacts (analogously to templates, or classes in OOP) which define the program and behavior of the concrete instances of those types. Each workspace is meant to have a (dynamic) set of artifact types that can be used to create artifact entities as instances. From the agent viewpoint, artifacts are the *first-class entities* structuring, from a functional point of view, the computational world where they are situated and that they can create, share, use, perceive at runtime.

To make its functionalities available and exploitable by agents, an artifact provides a set of *operations* and a set of *observable properties* (see Figure 4.2). Operations represent computational processes – possibly long-term – executed inside artifacts, that can be triggered by agents or other artifacts. The term *usage interface* is used to indicate the set of operations actually provided by an artifact at a specific time (the usage interface can change dynamically). Observable properties represent state variables whose value can be perceived by agents[1]; the value of

[1] Actually by those agents that are observing the artifact, as will be clarified later on.

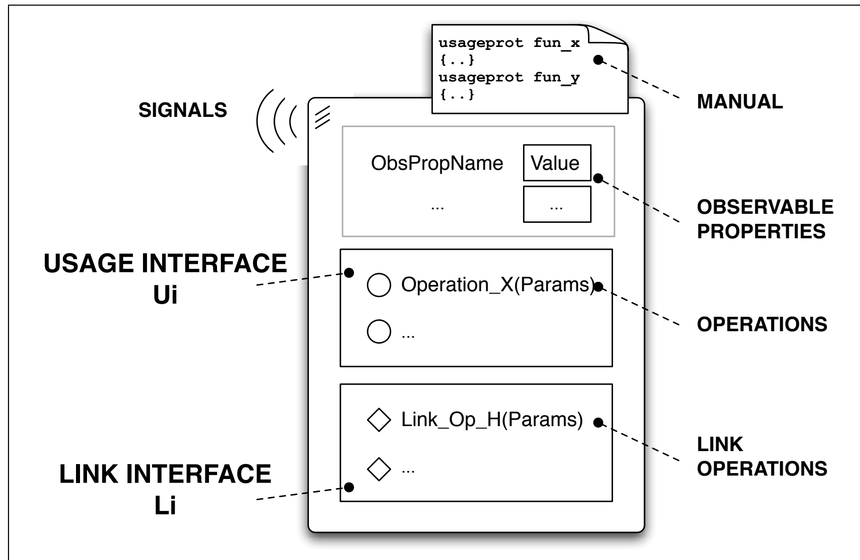

Figure 4.2: The abstract representation of an artifact in A&A model. In evidence the usage interface, the observable properties and the link interface.

an observable property can change dynamically, as result of operations execution. The execution of an operation can generate also *signals*, to be perceived by agents as well: differently from observable properties, signals are useful to represent non-persistent observable *events* occurred inside the artifact, carrying some kind of information. Besides the observable state, artifacts can have also an hidden state, which can be necessary to implement artifact functionalities.

From an agent viewpoint, artifact operations represent actions provided by the environment: so in artifact-based environments the repertoire of external actions available to an agent – besides those related to direct communication – is defined by the overall set of artifacts that (dynamically) populate the environment. Observable properties and events constitute instead agent percepts. To scale up with the environment complexity, in artifact-based environments an agent perceives the observable properties and events of only those artifacts that the agent intentionally (and dynamically) decided to observe.

As a principle of composition, artifacts can be *linked* together so as to enable one artifact to trigger the execution of operations over another linked artifact. To this purpose, an artifact can expose a *link interface* which, analogously to the usage interface for agents, includes the set of operations that can be triggered by

other artifacts—once that the artifacts have been linked together by agents, as clarified in next sub-sections. Linkability makes it possible to realize distributed environments, linking together artifacts belonging to different workspaces, possibly residing on different network nodes.

Finally, an artifact can be equipped with a *manual*, a machine-readable document to be consulted by agents, containing a description of the functionalities provided by the artifact developer and how to exploit such functionalities (that is, artifact *operating instructions* [140]). The information provided by the manual is meant to be dynamically read, interpreted and *internalized* by the agent, which is assumed to embed such a knowledge in terms of proper *plans* about how to use the artifacts of that type and when [118]. Such a feature has been conceived in particular for open systems composed by intelligent agents that dynamically decide which artifacts to use according to their goals and dynamically discover how to use them. An example of manual use will be described in Subsection 4.5.2.

4.2.2 Actions to Work with Artifacts

The set of actions available to agents to work with CArtAgO artifacts can be categorized in three main groups: *(a)* actions to create/lookup/dispose artifacts; *(b)* actions to use artifacts, executing operations and observing properties and signals; *(c)* actions to link/unlink artifacts. In the following, we informally describe these actions, while a formal description is provided in Chapter 5. The syntax $Name(Params) : Feedback_{\perp}$ is used to define action signature, which includes the action name, parameters and optionally the action feedback. The action feedback represents some kind of data – that depends on the specific action – which can result from the action execution, carrying information related to the success or failure of the action.

Creating and Discovering Artifacts

Artifacts are dynamic computational entities, which are meant to be created, discovered and possibly disposed by agents at runtime. This is a basic way in which the model supports dynamic extensibility (besides modularity) of the environment.

Three basic kinds of action are provided to this purpose: makeArtifact, disposeArtifact and lookupArtifact. makeArtifact(ArName,ArTypeName,InitParams):ArId instantiates a new artifact called ArName of type ArTypeName inside a workspace. The logic name identifies the artifact inside a workspace: artifacts belonging to

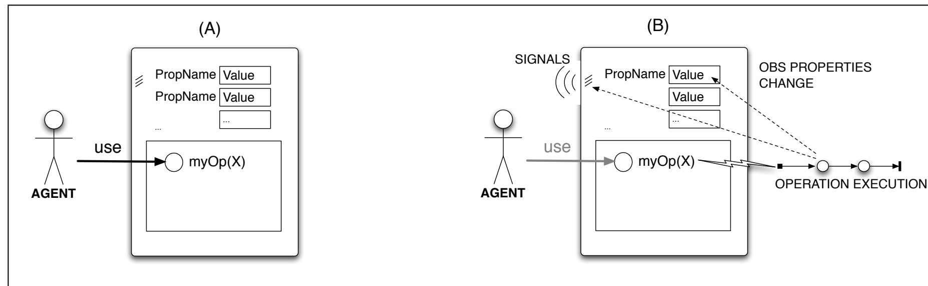

Figure 4.3: *(left)* An agent doing a use action to execute an operation listed in the usage interface of an artifact. *(right)* By executing the operation the observable property of the artifact can be changed and signals generated as observable events (for those agents observing the artifact).

different workspaces can have the same logic name, so besides the logic name each artifact has also a unique identifier generated by the system – returned as action feedback. Dually to makeArtifact, disposeArtifact(ArId) allows for removing an artifact from a workspace.

Artifact discovery concerns the possibility of retrieving the identifier of an artifact located in a workspace given either its logic name or its type description. A couple of actions are provided to this end: lookupArtifact(ArName):ArId which retrieves an artifact unique identifier given its logic name, and lookupArtifactByType(ArTypeName):{ArId} which retrieves the (possibly empty) set of artifacts that are instances of the specified type.

Using and Observing Artifacts

Interactions with artifacts, by an agent perspective, involve two main aspects: (1) being able to execute operations actually listed in the artifact usage interface and (2) being able to perceive artifact observable information, in terms of observable property and events.

For the first aspect, a single action use(ArId,Op):OpRes is provided (see Figure 4.3), specifying the identifier of the target artifact and the details about the operation to be executed (operation name and required parameters). The action succeeds if the triggered operation completes with success; conversely, the action fails if either the specified operation is not currently included in artifact usage interface or if some error occurred during operation execution and the process failed. By successfully completing its execution, an operation may generate some

result that are returned to the agent as action feedback.

This semantics makes it possible to consider artifact operations as agent actions, or rather the set of operations provided by artifacts as an extension of agents' action repertoire. Accordingly, since the operations executed by artifacts can be (long-term) processes, use actions triggering the execution of the operations can be long-term as well, not completing (with success or failure) immediately. As will be clarified in the formal model discussed in Chapter 5, the action-model adopted for agents to interact with artifacts is process-oriented. in Section 4.4 we will see how such a semantics could be exploited to create effective mechanisms for agents' action synchronization.

In the use action, further parameters can be optionally specified to refine the semantics and functionalities of the action: *(a)* a timeout, *(b)* an alignment predicate function, and *(c)* a tag. A *timeout* allows for specifying the time interval within the action (operation) must succeeds, otherwise it is considered failed—in spite of the result of the operation. The *alignment function* allows for specifying a condition over the observable state of the artifact which must be verified when the operation actually starts its execution, otherwise the action fails. This parameter is necessary when there is the need to enforce consistency between the environment state expected by the agent and artifact actual state when such action take place—expected state and actual state can be different because of the concurrency inside the MAS. Finally, by specifying a *tag* all the events generated by the artifact – and then the percepts on the agent side – will be annotated accordingly, by including the tag in the events. This can be useful on the observation side to select only those percepts that have been generated by a specific artifact, in the context of a specific operation execution.

For the second aspect, i.e. observation, an agent can start perceiving observable properties and signals of an artifact by doing a focus(ArId,{Filter}) action, specifying the identifier of the artifact to observe and optionally a filter to further select the subset of events the agent is interested in. An agent can focus multiple artifacts at the same time. Dually to focus, stopFocus(ArId) action is provided to stop observing an artifact. The perception model adopted is event-driven. In particular, by initiating a focusing activities, two types of percepts are received by agents:

1. percepts related to the updates of observable properties;

2. percepts related to signals sent by the artifact during an operation execution.

Thereby, every time an observable property is updated or a signal is generated

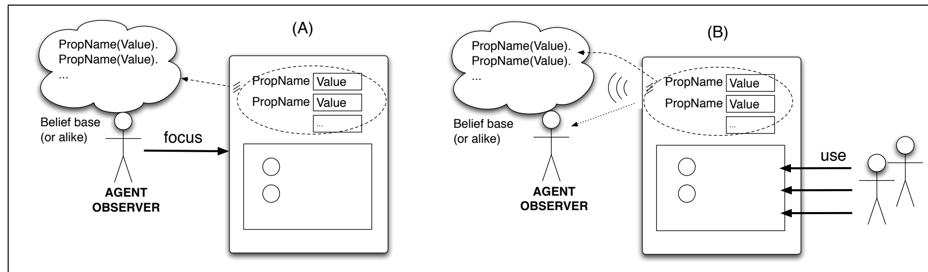

Figure 4.4: By doing a focus action on an artifact, an agent will eventually perceive the updated value of observable properties as percepts – mapped into agent beliefs or knowledge about the environment – and all the signals generated by the artifacts.

during an artifact operation execution, a related event is notified to all the agents which are focusing the artifact. Dispatching events is completely in charge of the CArtAgO platform, and the artifacts upon which the events are originated. From an agent perspective this allows to keep track of all environment changes, and, most important, ensures that no events will be lost even if the environment changes at a frequency which is greater than the one exhibited by the agent execution cycle.

It's worth noting that an agent can use an artifact also without focusing it. This may happen if the agent is not interested to the properties of the artifact or to the events that it generates. Conversely, if an agent executes an operation on an artifact that is actually focused, then the agent will receive all the percepts coming from operation execution (and while the use action has not completed yet). This allows agents to monitor the operation during their execution, possibly reacting once particular events occur.

We conclude the basic set of actions to use artifacts with the observeProperty(PropName):PropValue action, which synchronously reads the actual value of a specific observable property. In this case no percepts are involved and the value of the property is straightforwardly retrieved as action feedback. Differently from the focus case, this action is useful to *actively* inspect the environment, getting the value of an observable property when the agent needs it.

Inspecting Artifact Manuals

Inspecting artifacts manuals allow agents to internalize the usage protocols required to fulfill artifact functionalities. In order to fulfill this activity, the action

consultManual(ArtifactType) is provided. It allows agents to dynamically add to their programming model the instrumental knowledge (i.e., plans) required to exploit a given functionality. The parameter ArtifactType does not refer to the name of a specific artifact instance, but to the name of an artifact type, which class must be available in the current workspace.

Since a series of operative instruction, which are specified inside the manual, this action dynamically adds the module required to exploit the artifact functions (an example will be showed in Subsection 4.5.2). The added modules result as capabilities which can be mounted on the fly to improve the agent program [118]. Those capabilities are expressed by programming constructs typically given in the same language used to specify the agent. The dual action forgetManual(ArtifactType) allows agents to dynamically remove the module.

Linking and Unlinking Artifacts

Linking artifacts accounts for connecting two artifacts together so as to allow one artifact (the linking one) to execute operations over another artifact (the linked one). More precisely, by linking two artifacts the execution of an operation on the linking artifact may trigger the execution of operations on the linked artifact(s). To this end two basic actions are provided, linkArtifacts(LinkingArId,LinkedArId) and unlinkArtifacts(LinkingArId,LinkedArId), respectively to link and unlink two artifacts together. This allows either to an artifact to be *linked* by multiple artifacts, and to the same artifact to *link* multiple artifacts. This makes it possible for agents to dynamically compose complex artifacts by linking together simple ones, changing the links according to the current needs, creating networks of artifacts – which can distributed in different workspaces.

4.2.3 Actions to Enter and Leave Workspaces

In CArtAgO, agents can work simultaneously in multiple workspaces, eventually distributed among different network nodes. To work inside a workspace an agent must *join* it (and eventually *quit* as soon as it completed its work); The action joinWorkspace(WSId) allows agent to enter a workspace specified by an identifier, while the dual action quitWorkspace(WSId) allows agents to leave the specified workspace.

4.3 Environment Programming in CArtAgO

CArtAgO (Common ARtifact infrastructure for AGent Open environment) is a framework and infrastructure for programming and executing artifact-based environments implementing the model described informally above. As a *framework*, it provides Java-based APIs to program artifacts and the runtime environment to execute artifact-based environments, along with a library with a set of pre-defined general-purpose artifact types. As an *infrastructure*, it provides the underlying mechanisms to extend agent programming languages / architectures as specified in Table 4.1.

To give a taste of artifact programming, in the following we provide an informal description of some main aspects concerning the API: further details can be found in the documentation available in CArtAgO open-source distribution [20].

4.3.1 Artifact Programming Model

The Java-based API allows for programming artifacts in term of Java classes and basic data types, without the need of using a new special-purpose language to this end. In the following we give an overview of the basic features of the programming model using some simple examples.

An artifact (type) is programmed directly by defining a Java class extending the library class `alice.cartago.Artifact`, and using a basic set of Java annotations and inherited methods to define the elements of artifact structure and behavior[2]. The type defines the structure and behavior of the concrete instances that will be instantiated and used by agents at runtime. Figure 4.5 shows a simple example of artifact type definition implementing a simple counter (a `Counter` artifact). The counter artifact provides a single observable property called `count` keeping track of a count value, and an operation `inc` exploitable by agents to increment the value of the count. The `init` method is used by convention to specify how the artifact must be initialized at creation time—if the method generates an error, the artifact is not created and the agent action fails. Observable properties are defined by means of the `defineObsProperty` primitive specifying the name of the property and the initial value (which can be any tuple of data objects).

[2] The annotation framework is a feature introduced with Java 5.0 that makes it possible to mark some elements of a class description on the source code – including methods, fields, and the class definition itself – with some descriptors that can be accessed at runtime by the program itself, through the reflection API. Annotations are represented by symbols of the kind `@ANNOT_NAME`, possibly containing also attributes `@ANNOT_NAME(attrib=value,...)`.

```
class Counter extends Artifact
{
  void init(){
    defineObsProp("count",0);
  }

  @OPERATION void inc(){
    int c = getObsProperty("count").intValue();
    int newc = c + 1;
    updateObsProperty("count", newc);
  }
}
```

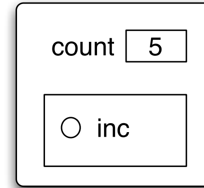

Figure 4.5: *(left)* Definition of a simple `Counter` artifact type. It exposes a usage interface with a single operation `inc` and an observable property `count`. *(right)* An abstract representation of the `Counter` artifact, with in evidence the usage interface and the observable property.

Other two primitives are available to retrieve the current value of the property (`getObsProperty`) and to change its current value (`updateObsProperty`). Instance fields of the class are used to implement artifact internal non-observable state (no internal variables are defined in `Counter`).

Operations are defined by methods annotated with the `@OPERATION` tag and `void` return value, using method parameters as operation request parameters. A single method is sufficient to implement simple operations, i.e. operation composed by a single *atomic* computational step. It is the case of the `inc` operation in the `Counter` artifact, which simply increments the value of the observable property `count`.

In a more complex case, necessary for instance to implement long-term operations whose execution can be controlled by agents through percepts, operations can be structured, namely composed by multiple (atomic) steps. In multi-step operations, `@OPERATION` method represents the first atomic step to be triggered by a user agent. Further atomic steps are launched by the operation itself, and are implemented by methods annotated with `@OPSTEP`. For triggering new operation steps inside an operation the `setNextOpStep` primitive is used, and it indicates as a parameter which is the next step to be executed. The execution of a multi-step operation completes when no further steps have to be executed. Each step is executed atomically in a chined sequence of operation steps. It is worth remarking that only a single operation step can be in execution inside an artifact at a time,

```
class ArtifactWithSteps extends Artifact
{
  int value;
  boolean isAvail;

  @OPERATION void getNewValue(){
    isAvail = false;
    setNextOpStep("processValue");
  }

  @OPSTEP(guard="isValueAvail")
  void processValue(){
    setOpResult(value);
  }

  boolean isValueAvail(){
    return isAvail;
  }

  @OPERATION void setValue(int v){
    value = v;
    isAvail = true;
  }
}
```

```
class Counter2 extends Artifact
{
  boolean counting;

  void init(){
    counting = false;
  }

  @OPERATION(avail_when="isIdle")
  void start(){
    counting = true;
    triggerOp("counting");
  }

  boolean isIdle(){ return !counting; }

  @OPERATION(avail_when="isCounting")
  void stop(){
    idle = true;
  }

  boolean isCounting(){ return counting; }

  @INTERNAL_OPERATION
  void counting(){
    setNextOpStep("stepCount");
  }

  @OPSTEP void stepCount(){
    if (counting){
      signal("tick");
      setNextOpStep("stepCount");
    }
  }
}
```

Figure 4.6: *(left)* An artifact providing a multi-step operation `getNewValue`: the second step is executed as soon as a value has been inserted (by means of `setValue` operation). *(right)* An example of an artifact with an internal (multi-step) operation (`counting`), whose execution is triggered and controlled by operations which are part of the usage interface (`start` and `stop`).

in order to avoid the concurrent modification of artifact internal state. But, while each single-step is always executed in a mutually exclusive way, multi-step operations can be executed concurrently, by interleaving each step. Given this, if an agent tries to execute an operation included in the usage interface in which a step is in execution, then a new instance of the operation is triggered but its execution *suspended* until the step has completed.

For each step, including the first one annotated with `@OPERATION`, a guard can be specified to define the condition that must hold to execute the step, once it has been triggered. Guards are defined by means of the `guard` attribute in `@OPSTEP` and `@OPERATION` annotations, specifying the name of a boolean method implementing the guard. An example of artifact with multi-step operations with guards is shown in Figure 4.6 (left). In this case, the operation `getNewValue` in `ArtifactWithSteps` has two steps, the second one (`processValue`) with a guard (`isValueAvail`) so that the step is executed (after being triggered) only when a value has been inserted (by means of the `setValue` operation). As showed in [119], Multi-step operations and guards are useful in particular to program coordination artifacts, which are assumed to synchronize the concurrent execution of multiple operations inside the same artifacts.

By completing an operation – either single or multi-step – a result can be specified by means of the `setOpResult` primitive; the result is retrieved by the agent as action feedback. If no result is specified, the feedback is not defined. An example of operations producing a result is the multi-step `getNewValue` operation of `Counter2`, as shown in Figure 4.6 (right). Analogously, in the case of operation error – and then use action failure – the `setOpError` can be used to specify the kind of error.

Operations can themselves trigger the execution of *internal* operations, that are operations which are not included in the artifact interface, thus not visible from external agents neither artifacts. The definition of internal operations is the same of normal operations but the annotation to be used, which is `@INTERNAL_OPERATION`. To trigger the execution of internal operations the `triggerOp` primitive is provided, requiring the name of the operation and the parameters. The `Counter2` artifact in Figure 4.6 has an internal multi-step operation `counting` which is triggered by the `start` operation. The operation `counting` repeatedly executes a step until the `stop` operation is executed. It's worth remarking that `triggerOp` does not execute directly the specified operation (as a method call): it just triggers its execution, which actually starts as soon as the overall artifact configuration allows that, namely no more than one operation step in execution at the same time inside the same artifact. This example also shows the implementation of artifacts with

```
class LinkedArtifact extends Artifact
{
  ...
  @LINK void linkedOp(int param){
    ...
  }
}
```

```
class LinkingArtifact extends Artifact
{
  ...
  @OPERATION void myOp(int x){
    ...
    triggerLinkedOp("linkedOp",x);
    ...
  }
}
```

Figure 4.7: *(left)* An artifact exposing an operation (`linkedOp`) which can be triggered by other linked artifacts; *(right)* An example of an artifact with an operation triggering the link operation by means of the `triggerLinkedOp`

a *dynamic usage interface*, where the set of available operations in the usage interface depends on the state of the artifact. To this end, a `avail_when` attribute in `@OPERATION` can be specified, using a boolean function to represent the condition that must hold on the artifact state so as to include the operation in the usage interface. In the example the `stop` operation is part of the usage interface only if a counting process is active inside the artifact.

By executing an operation the artifact can generate *signals* that will eventually be perceived as observable events by agents focussing the artifact. Signals can be generated by the `signal` primitive, specifying the type of the signal and optionally an information content (which can be a tuple of values). In `Counter2` artifact shown in Figure 4.6, a `tick` signal is generated each time a `stepCount` operation step is executed and the counting process has not been stopped.

Being based on the Java platform, the environment data model adopted in CArtAgO is based on the Java object model: so the data types used in operation parameters, observable properties and signals are either Java primitive data types or objects instances.

Finally, linkability is supported by properly annotating with `@LINK` marks those operations of an artifact that can be linked by other artifacts. The operations annotated with `@LINK` are included only in the *link interface* of the artifact. Link operations of an artifact are not visible by agents but only by other artifacts which have been linked to the artifact itself. In so doing, linking artifacts can trigger the execution of a link operation listed in the link interface of a linked artifact. The primitive to be used inside an operation step in order to trigger operations in external linked artifacts is `triggerLinkedOp`. It requires the name and the parameters of the operation to be triggered. In the example shown in Figure 4.7, the artifact `LinkedArtifact` provides a `linkedOp` link operation, which can be triggered as shown in the `myOp` operation of `LinkingArtifact` artifact. It's worth remarking

```
(1)  use(ArId,Op):OpRes
(2)  focus(ArId,{Filter})
(3)  stopFocus(ArId)
(4)  observeProperty(PropName):PropValue
(5)  makeArtifact(ArName,ArTypeName,InitParams):ArId
(6)  disposeArtifact(ArId)
(7)  lookupArtifact(ArName):ArId
(8)  lookupArtifactByType(ArTypeName):{ArId}
(9)  linkArtifacts(LinkingArId,LinkedArId)
(10) unlinkArtifacts(LinkingArId,LinkedArId)
(11) joinWorkspace(WSId)
(12) quitWorkspace(WSId)
```

Table 4.1: Basic repertoire of actions to work in CArtAgO environments.

that an artifact can trigger the execution of a link operation over another artifact if and only if such artifacts have been previously linked by an agent, by means of the linkArtifacts action.

4.3.2 Integration with Agent Programming Platforms

CArtAgO programming model is orthogonal to the specific technology adopted for programming the agents playing within artifact-based environments. The CArtAgO technology has been conceived so as to be integrated with any agent programming language and platform, so as to create heterogeneous systems where agents – possibly implemented using different agent programming languages and technologies, and running on different platforms – could work together in the same MAS, sharing common artifact-based environments [117].

Technically, by integrating an agent programming language/framework with CArtAgO, the repertoire of agent actions has to be extended with the new set of actions, as they have been discussed in previous section and resumed in Table 4.1. On the perception side, the set of possible agent percepts are extended with observable properties and signals generated by artifacts. Thanks to integration technologies observable properties are mapped into agent beliefs (or knowledge, if the notion of belief is not supported) about the state of the environment (artifacts), instead signals as beliefs about the occurrence of observable events.

The concrete realization of the integration technologies can vary, depending on the specific agent programming platform to be integrated. Actually, along with

CArtAgO a set of bridge technologies are available for integrating *Jason*, *Jadex*, simpAagent models [117, 109].

4.4 Agents at work in CArtAgO Environments

To give a concrete taste of programming model, here we briefly describe a simple examples of programming agents and artifacts in CArtAgO environments. In what follows we first briefly resume some fundamental notions of agent programming language in *Jason* ([14, 69]). Besides *Jason*, that will be used as the reference language in the following of the work, other agent languages have been integrated in CArtAgO. In [109] the interested readers can find examples of *Jadex* integration [112, 68], while in [123] CArtAgO is integrated with a Java-based activity-oriented agent framework called simpA.

4.4.1 Agent Programming in *Jason*

To ease the understanding of the agent source code that will be used in the following of this work, we here report a brief description of *Jason* syntax. An agent program in *Jason* is defined by an initial set of beliefs, representing agent's initial knowledge about the world, a set of goals, and a set of plans that the agent can dynamically instantiate and execute to achieve such goals. The syntax of AgentSpeak is based on the notion of plans. A plan is triggered by some event and is guarded by some context, the syntax is:

```
<event> :  <context> <- <body>.
```

In the case where the event happens and the context holds, the plan body is executed. In particular:

- `<event>` represents the specific event triggering the plan – examples are the addition of a new belief (`+b`), a goal (`+!g`), the perception of an observable event generated by an artifact (`+ev [source(?Art)]`), the perception of an update of an artifact observable property, (`+p [artifact(?Art)]`).
- `<context>` is a logic formula on the belief base – a belief formula – asserting the conditions under which the plan can be executed.
- `<body>` includes a list of basic actions exploitable, for instance, to query the belief base (`?g`), create subgoals to be achieved (`!g`), to update agent inner

state – such as adding a new belief +b – to send messages and ACL to other agents, to verify constraints, etc.

For instance, the plan:

```
+count(V) [artifact("c1")]
: bel(X) & X > V
<- action1; action2; ...
```

is triggered when the agent perceives from a counter artifact called c1 a given count value V (+ means that something was added to the belief base, as a perception in the above example).

If the agent can prove from its beliefs that bel(X) & X > V holds, then the plan is selected for execution and becomes an intention. Finally, the intention is executed by performing the specified actions of the body.

The set of available actions in *Jason* can be easily extended by means of internal actions. We thus create a library of internal actions improving agent repertoire and implementing all the actions needed to interact in CArtAgO environments. CArtAgO actions – which are prefixed by "cartago." – include, among others, use, to execute an operation on an artifact, focus, to start observing a specific artifact, makeArtifact to create a new artifact, and lookupArtifact to get artifact unique identifier given its name. Further and more detailed examples of *Jason* agents working in CArtAgO environments are showed in the next sections.

4.4.2 Using simple Artifacts

As a simple example, Figure 4.8 shows two agents programmed in *Jason*, working in the same workspace and using CArtAgO API to use and observe artifacts: the first one (on the left) creates and uses two artifacts of type Counter and Counter2, defined in Subsection 4.3.1, and the second one (on the right) locates and observes the artifacts, reacting to related percepts. A more detailed description of the source code follows. The first agent has a single initial goal create_and_use. A plan to achieve the goal is specified, which is triggered by the +!create_and_use goal addition event. The plan creates an instance of Counter and Counter2 artifacts called c1 and c2 and uses them. First it starts the counting process on c2 by executing the start operation; then it executes the inc operation on c1, and finally it reads the value of the observable property count by means of the observeProperty action and prints it on standard output be using the console

```
// user agent

// initial goal

!create_and_use.

// plans

+!create_and_use : true <-
  cartago.makeArtifact("c1","Counter",C1);
  cartago.makeArtifact("c2","Counter2",C2);
  // start the counting on c2 counter
  cartago.use(C2,start);
  // use c1 counter
  cartago.use(C1,inc);
  cartago.observeProperty(C1,count(V));
  cartago.use(console,
     println("Count value: ", V)).
```

```
// observer agent

// initial goals

!discover_and_observe("c1").
!discover_and_observe("c2").

// plans

+!discover_and_observe(ArName): true
 <- cartago.lookupArtifact(ArName,C);
    cartago.focus(C).

+count(V) [artifact("c1")] : V < 10
  <- cartago.use(console,println("count:  ",V)).

+count(V) [artifact("c1"), artifact_id(C)] :
  V >= 10
  <- cartago.use(console,println("count:  ",V));
     cartago.stopFocus(C).

+tick [source("c2")] : true
  <-  cartago.use(console,println("tick. ")).
```

Figure 4.8: *Jason* (AgentSpeak) agents at work with artifacts: on the left an agent creating and using two artifacts of type `Counter` and `Counter2`, on the right an agent observing the two artifacts.

artifact[3]. The second agent has two initial goals `discover_and_observe`, one for each artifact instance. The plan to discover and observe the specified artifact simply accounts for using `lookupArtifact` to locate the artifact and then `focus` to start observing it. In *Jason* by focussing an artifact, observable properties are mapped into beliefs, annotated with `source(percept)` annotation, which are automatically updated as soon as the value of observable properties changes. Also signals are mapped onto beliefs, annotated with `source(ArtName)`. Three plans are specified to react to percepts related to the two artifacts. As soon as the agent perceives a new value for the `count` observable property (in `c1` artifact), it prints a message about the new value on standard output by exploiting the `console` artifact. If the perceived value is greater or equal to ten, then the agent stops observing the artifact. Then, as soon as the agent perceive a `tick` signal (generated by `c2`), it prints a message on the console too.

[3] An instance of `console` artifact is available in each workspace and it is useful to print messages to standard output

```
public class Agenda extends Artifact
{
    ...

  @OPERATION
  void schedule(String todo, long when){
    setNextOpStepTimed(when, "alarm", todo);
  }

  @OPSTEP void alarm(String todo){
    signal(todo);
  }
}
```

```
/* Jason agent exploiting the agenda */
...
+!setupAgenda :
  task1_date(T1)
  & task2_date(T2)
  <- cartago.makeArtifact("ag","Agenda",A);
     cartago.focus(A);
     cartago.use(A, schedule("task1", T1));
     cartago.use(A, schedule("task2", T2)).

/* my scheduled activities*/

+task1[artifact("ag")]: true
<-     // activities related to task1

+task2[artifact("ag")] : true
<-     // activities related to task2
```

Figure 4.9: A sketch of a personal agenda artifact and a usage example by a *Jason* agent.

4.4.3 Using Artifacts to Externalize Activities

Externalization is a typical outcome once agents are capable to coordinate their activities with external resources. Figure 4.9 shows a typical example of externalization of agent activities on artifacts. The example assumes a simple *personal agenda* artifact that can be used by an agent to annotate (schedule) tasks to do in the future. The artifact will eventually generate a signal (alarm) at the specified time, so as to allow the agent to react accordingly and fulfill the programmed task.

In general, agents externalizing activities on artifacts succeed at: *(i)* extending action repertoire without the need to extend architectures/languages; *(ii)* reducing the computational burden—agents do not waste time and computational resources for the execution of the operation and processes related to the externalized functionalities, which are instead executed inside artifacts; *(iii)* enhancing reusability—tools (artifacts) can be flexibly re-used among heterogeneous agents, even developed with different agent programming languages. *(iv)* dynamic extensibility—tools can be created/disposed at runtime by need.

4.5 Cognitive Use

An important aspect of agent-artifact interaction can be envisaged when artifact functionalities are employed in the context of societies of cognitive agents, namely agents capable to reason about their epistemic and motivational states. This sec-

tion resumes remarkable aspects by describing examples.

4.5.1 Mapping Goals and Beliefs on Artifact Functions

A twofold kind of interaction has been described in terms of cognitive use of artifacts [105]. On the one hand, artifact representational functions[4] has been assumed to improve agents epistemic states, i.e., by representing and providing strategic *knowledge* in the overall system (artifact *epistemic function*). On the other hand, artifacts operational functions allows agents to improve the repertoire of their actions, i.e., by providing additional *means* which can be purposively used to achieve goals (artifact *operational function*).

A producers-consumers scenario is here resumed to introduce the twofold approach. The producers-consumers problem is typical in concurrent systems, where agents are supposed to adopt effective strategies with respect of the shared resource and taking into account further bounded resources like time and space (memory). This requires some kind of coordination strategy between agents, i.e., in order to coordinate the cyclic production of items by producer and the activities performed by consumer agents. The example makes use of a *Bounded-inventory* artifact, that, besides main aspects of the artifact programming model as observable properties and a usage interface, introduces synchronization functionalities. The bounded-inventory is a kind of coordination artifact designed to function as a shared inventory mediating the exchange of some kind of *item* between a possibly dynamic number of *producer* agents and *consumer* agents [82]. In particular the action of a consumer retrieving an item from the buffer is suspended until at least one item is available. Viceversa, the action of a producer inserting an item is suspended whether the buffer capacity is full. Finally, bounded-inventory artifacts can be programmed as regulatory mechanisms, i.e., by balancing the capacity of the inventory, and further adopted for tuning the global performance of the system.

Looking at the CArtAgO implementation of the bounded-inventory artifact (Figure 4.10), it provides a usage interface with two operation controls to respectively insert (`put`) e consume (`get`) items, and two observable properties, `max_nitems`, showing the maximum capacity of the inventory, and `n_items`, showing the current number of items stored in the inventory. Internally, a simple linked list is used to store items. The synchronization functionality provided by the artifact is realized here by exploiting a basic feature of the artifact programming

[4]The term "function" here must be interpreted as "functionality", so not related to functional programming languages or mathematical functions.

```
public class BoundedInventory extends Artifact
{
  private LinkedList<Item> items;
  private int nmax;

  void init(int nmax){
    items = new LinkedList<Item>();
    defineObsProperty("n_items", 0);
    this.nmax = nmax;
  }

  @OPERATION(guard="bufferNotFull")
  void put(Item obj){
    items.add(obj);
    int ni = items.size() + 1
    updateObsProperty("n_items", ni);
  }

  boolean bufferNotFull(Item obj){
    return items.size() < nmax;
  }

  @OPERATION(guard="itemAvailable")
  void get(){
    Item item = items.removeFirst();
    int ni = items.size()-1;
    updateObsProperty("n_items",ni);
    setOpResult(item);
  }

  boolean itemAvailable(){
    return items.size() > 0;
  }
}
```

Figure 4.10: Implementation of an artifact functioning as a bounded inventory in producers-consumers scenario.

model, which accounts for the possibility of defining *guards* that specify when an operation is either enabled or disabled. In the example the `put` operation is allowed only when the inventory is not full (`bufferNotFull` guard is true), and `get` is allowed when the inventory is not empty (`itemAvailable` guard is true). Thanks to the specified guards, if an agent triggers the `put` operation when the inventory is full, the action is suspended. Analogously, the action is suspended when the inventory is empty and a `get` operation is triggered.

Purposive function

As far as the bounded-inventory artifact has been conceived, the provided operations, that may be controlled by artifact usage interface, encapsulate artifact's intended purposes[5]. Indeed, from an agent viewpoint, operations can be suitably used to achieve goals, namely provide and retrieve items. In this case in particular, operations can be dynamically triggered by agents so as to *externalize* and distribute (part of) their goal-oriented activities. For doing this, operation outcomes have to be taken into account by agents in their practical reasoning. In fact, by changing the actions required for achieving a given goal, artifact operations

[5]Notice that before being in the intention of an agent who wants to use the artifact, the intended purpose is in the mind of artifact designer, who conceive it in order to serve an operation or a function.

change agent means-end reasoning[6] stages.

This aspect can be tackled at different conceptual levels. The first, most obvious, solution is to integrate artifacts functionalities during agent's developing phases. In this view, artifacts purposive use can be defined at the language level, by defining the operation to use in an "off-line" fashion, namely at design time. Such an approach envisages agents goal in the operation outcomes, e.g, allows agents to achieve goals by the mean of operations which have been defined – by the artifact developer – within artifact control interface. Referring to the bounded-inventory example, an agent having the goal to produce a new item and put it in the buffer may use the following intention (agent's specification is provided with *Jason* language):

```
+!produceItems : nextItemToProduce(Item)
  <- cartago.lookupArtifact("my-inventory"
             , InvID)
     cartago.use(InvID, put(Item)).

-!produceItems: true
  <- cartago.use(console,
        println("Insertion failed")).
```

The agent in this case selects the intention to store an item on the inventory once an `Item` has been prepared and is available in the belief base. Then the adopted plan first lookups the `my-inventory` artifact to retrieve its system identifier `InvID` and then stores the item by selecting the `put` operation provided by the artifact usage interface. Notice here the presence of a fail handling plan (`-!produceItems`) which is triggered whether the insertion fails in order to write a message on the console.

Besides, a consumer agent can cyclically adopt the following plans to attain an item on the inventory:

```
+!consume
  : myInventory(InvID)
  <- cartago.use(InvID, get, Item);
     !consumeItem(Item);
     !!consume.

-!consume
  <- // handle failure ...

+!consumeItem(Item)
  : true
  <- // process Item ...
```

Notice in this case that the consumer agent simply maps the execution of its external action on the `get` operation provided by the buffer artifact. In so doing, the item is retrieved as an action feedback as the `get` operation completion state.

[6] We here refer to the notion of cognitive agents able to find a suitable sequence of actions, between the ones he has in repertoire, so to attain an adopted goal. Several agent architectures founded on this reasoning principle have been presented in the last years, many of which can be related to the conceptual model provided by [16].

Generalising the artifact purposive function, once included in agents repertoire of actions, artifact operations are assumed to improve agent repertoire of actions, thus providing additional means for agents to achieve their goals.

a final remark is worth to be taken into account on the purposive function of artifacts. A first pivotal aspect in treating artifact operational functionalities relates on the specific motivational attitudes adopted by agents. Actually, the abilities to handle goals are variously characterized by mainstream agent platforms [136]. The procedural goal approach can be related to agents functioning according to transitions within their action selection mechanisms, while typically the goals are not not explicitly represented in agents specification and where a behavioral strategy is rather specified by the programmer, through the composition of procedures taking into account the intended goal states. In the case of agents adopting procedural goals, namely without the possibility to map goal representations on operation outcomes, to a *goal-oriented* use of artifacts. On the other hands, declarative goal approaches refer to agents able to process goals which are explicitly represented as internal states. In this case declarativeness stands for explicit representation of goals described either in terms of end-states, either in terms of execution states within the reasoning process. As discussed in [109] describing integration between the *Jadex* agent platform and CArtAgO, we refer in this case to a stronger notion of goal, thereby at the basis of *goal-directed* use of artifacts.

Epistemic function

A second function, dual to purposive one, is about informational, observable and readable knowledge provided by artifacts and represented by observable properties. In this case, from an agent point of view, artifacts can be viewed as informational units functioning to maintain, make it observable and, possibly, pre-process information which is relevant for agents knowledge base. In other terms, by embedding machine-readable representations, an artifact can be a target for agents epistemic actions[7]. This entails for agents the opportunity to read and observe artifacts to attain new information and possibly update beliefs, solely with the aim to improve the knowledge base with information which is strategic for the fulfillment of their tasks. In this view, artifacts are supposed to provide observable cues in order to highlight relevant information (thus improving agent's situated cognition). This turns to be important for shaping *goal-supporting beliefs*, i.e.

[7]The notion of epistemic action adopted here refers to the one introduced by D. Kirsh and P. Maglio to indicate those action aimed at improving agents knowledge [78]

those beliefs required to agents for ruling over deliberation and practical reasoning [26]. Accordingly, information available with observable properties can ease agent reasoning, for instance simplifying and improving agent's decision making and remarkably easing belief update processes.

As a simple example of epistemic use, we consider here an extension of the producer-consumer scenario where two bounded inventories are deployed instead of one. By introducing an additional inventory it is possible to avoid centralization and bottlenecks during agent activities. In this view, we assume that, by continuously observing the number of items of both the inventories, consumer agents can dynamically decide which artifact to use according to some utility value. By considering that the probability to get stuck is minimized for consumers when the inventory is not empty, we assume consumers choosing the inventory with more items. To this end the continuous observation of the `n_items` observable properties on both the inventories is performed through a focus action:

```
+!consumeActivity  : true
  <- +min_items(-1);
     cartago.lookupArtifact("my-inventory-1",
           InvID1);
     cartago.focus(InvID1);
     cartago.lookupArtifact("my-inventory-2",
           InvID2);
     cartago.focus(InvID2);
     +selectedInv(InvID1,0);
     !!consumeAction.
```

```
+n_items(N)[artifact_id(InventoryID)]
  : selectedInv(_, N1) & N > N1
  <- -+selectedInv(InventoryID, N).

+!consumeAction : selectedInv(InvID, _)
  <- cartago.use(InvID, get, Item);
     cartago.use(console,
           println( " Consumed Item: ", Item));
     !consumeAction.
```

The agent here uses a unique goal-supporting belief `selectedInv(InventoryID,NItems)` to store the identifier of the inventory, among the observed ones, with the greatest number of items. Such a belief is initially set specifying the `my-inventory-1` artifact in the `consumeActivity` plan. As soon as a new value of the observable property `n_items` is perceived, the belief is updated, storing from time to time the artifact which has the greatest amount of items to consume. The plan annotation `[artifact_id(InventoryID)]` makes it possible to retrieve the identifier of the artifact from which the percept raised. In this case agents are aware of the current state of the artifacts which are *perceived* since observable properties updates. Focusing the inventory allows the agents to translate on the fly the events coming from the artifact. Once a percept indicating the property update is received, the agent straightforwardly react to the event by resuming the suspended `consumeAction` intention and update his goal-supporting beliefs. A similar strategy can be implemented for the producer agents (the code is here omitted for brevity) that can use a twofold strategy for choosing the inventory where to put a

new item.

An important aspect to take into account in cognitive interactions is the capability for agents of controlling activities in order to monitor the externalized processes. This is what Norman refers, in the human case, as “gulf of execution and evaluations” of activities upon artifacts, by which an individual may manage the interaction, i.e. evaluating signals between his expectations and the actual course of observable events [87]. Working on the perception model based on perceiving signals coming from ongoing operation, external events coming from an artifact have been integrated at an architectural level by automatically promoting such events as internal signals to be filtered to update beliefs. Otherwise, once encoded, the events controlling a given interaction can bypass the deliberation phase (or intention selection) and can be addressed by routinized activities realizing reactive behaviors, as seen in [109]. Even more, by introducing mismatch-based filtering rules, particular events can be used for signalling critical situations requiring servicing and for triggering the adequate responses.

Besides 1 to 1 interactions, the contribute of artifacts in easing agent epistemic activities is remarkable also in the context of Multi Agent scenario. Once societies of agents are of concern, the pivotal aspect is the availability of information in the overall society of agents. As seen in [106], information can be spread over several orthogonal dimensions: *(i)* across agents: by organising and making available relevant information as permanent side-effect of artifact use (modification of artifact state); *(ii)* across platforms: once interactions between agents are mediated by artifacts, heterogeneous platforms can be integrated at the same domain level. Moreover agents acquire an additional option to communicate, being artifacts a suitable alternative to protocols based on message exchange; *(iii)* across time: artifacts are designed to hold strategic information which can persist also over interleaved presence of individual agents; *(iv)* across space: the topological notion of work environments makes it possible for agents to distribute their activities between many nodes and workspaces. This entails no need for agents mutual presence within a given location/place.

4.5.2 Externalisation and Internalisation

Besides “off-line” interactions which are specified at a programming level by agent developers, an alternative approach can be envisaged to exploit artifacts through an “on-line” integration of their functionalities. In this approach agents are capable to dynamically discover and afford artifact which are *not known* at design time, in so doing *externalising* part of their activities into the services

```
usageprot compute_sin {
  :function sin(X,Y)
  :body {
     locateMyTool(ToolId);
     freshSensor(S);
     use(ToolId,computeSin(X),S);
     sense(S,sin(X,Y)).
  }
}


package tools;

public class Calculator extends Artifact
{
  ...
  @OPERATION void computeSin(double x){
    signal("sin",x,Math.sin(x));
  }
  @OPERATION void computeCos(double x){
    signal("cos",x,Math.cos(x));
  }
  ...
}
```

```
// Jason Agent Internalising Calculator functions

!doComputations
  <- !setup;
     !doTheJob.

+!doTheJob
  <- cartago.consultManual("tools.Calculator");
     cartago.consultManual("tools.Console").

+!doTheJob
  <- !sin(1.57,Y);
     !print("The sin value of 1.57 is ",Y).
```

Figure 4.11: (Left) A usage protocol defined in the `Calculator` manual and the CArtAgO implementation of the `Calculator` artifact type. (Right) *Jason* agent exploiting the manual to use the `Calculator`

provided by the computational environment where agents are situated. In order to enable agents (and agent programmers) to exploit artifact functions, the dual notion of *internalising* artifact functions has been introduced, which consists in dynamically consulting and automatically embedding high-level usage protocols described in artifact *manuals*. This approach requires the additional capability for agents to scrutinize artifacts and map their functions on their internal goal base and has been presented in [118]. Two actions are provided respectively for internalising and forgetting the content of a manual: `consultManual(ArtifactType)` and `forgetManual(ArtifactType)`. By consulting the manual, the practical knowledge contained inside is fetched and translated into agent local plans. Those plans augment the repertoire of agent actions, and, in the case of goal oriented / directed agents, can be triggered by achievement goals which have the same signature of the artifact function retrieved on the manual.

A simple first-order logic-based language is used to define the manual protocols: the complete syntax and semantics of the language is not reported here due to lack of space, we informally describe the language by means of a concrete example. Figure 4.11 shows a usage protocol defined in the manual for the a

`Calculator` artifact. The `Calculator` artifact provides mathematical functions (i.e. *sin, cos, sqrt*, etc.) which an agent may exploit in a purposive fashion.

In the artifact manual, the function is specified by means of `:function` tag and is represented by a logic term, possibly containing parameters detailing input and output (in terms of unbounded variables) and information characterizing the function. In the calculator example, `sin(X,Y)` is the function of the usage protocol to compute the sine function. The function of a usage protocol is directly mapped to agent goals: in particular, a usage protocol with a function $func$ is mapped into agent plan(s) that are triggered to achieve goals matching $func$, according to some kind of matching function that depends on the agent architecture adopted. In the case of *Jason* agents, for instance, the usage protocol is triggered to achieve goals of the type `sin(X,Y)`: in the example (Figure 4.11, on the right) this happens by means of the `!sin(1.57,Y)` action.

The condition under which the functionality can be exploited can be specified by the `:precond` tag and is represented by a logic expression specifying the context conditions that must hold concerning either the function parameters or agent beliefs[8] (which typically can include the state of the observable properties of the artifact). If missing, the default value of the precondition expression is `true`.

The body – specified by means of the `:body` tag – contains a sequence of actions, including basic CArtAgO actions (`use`, `observeProperty`, `focus`, etc.), auxiliary actions to locate artifacts and internal actions for inspecting and updating the belief and goal base of the agent. From a syntactical point of view, `;` is used as sequence operator, `+Bel` and `-Bel` are used to add and remove beliefs and `.` to indicate the end of the plan.

The key point here is that the agent programmer has not to be aware and explicitly code the usage protocol, which is specified – instead – by artifact developers: s/he must simply know the interface of the usage protocol, in terms of the function and beliefs involved. Finally, the approach promotes a strong separation of concerns and finally more compact and readable agent programs.

[8]The notion of "belief" can be replaced here with "knowledge" for agent programming languages not having that concept

4.6 Final Remarks on programming Agents and Artifacts

This chapter described the basic features of CArtAgO, a platform for building working environments based on the A&A model. The choice of an A&A approach to MAS envisages agents as the basic abstraction to design and program the autonomous part of the systems, i.e. those parts that are responsible of the autonomous execution of some kind of task. On the other hand, artifact is conceived as the abstraction to design and program the functional part of the system, namely that part that can be exploited and controlled at run time for easing agents' activities. The notion of workspace is introduced to group together coherent set of agents and artifacts, typically devising the bounds of an application domain.

CArtAgO provides a programming platform and a run time support for building distributed work environments, which can be structured in several workspaces spread across different nodes. The independence between artifacts and agents computational models allows the reification of open systems, where heterogenous agents may join and leave without particular requisite on their architectures. The only requirements are those related on the capabilities needed for agents to interact inside artifact based work environments. The interaction is thus enabled since the definition of a basic set of actions extending agents' repertoire with the actions needed to operate within artifacts and workspaces.

By integrating CArtAgO with existing agent computational models platforms, a new degree of "separation of concerns" is introduced with environment programming: programming agents on the one side, so as to encapsulate autonomous and pro-active activities, and programming artifacts on the other side, representing those resources and tools that will be instantiated, shared and used by agents at runtime. The A&A approach is thus meant to support the development of *heterogeneous* MAS, composed by agents with different computational models – from reactive up to cognitive ones – cooperating in the same work environment.

The last part of the chapter introduced agent artifact interaction once a stronger notion of agency is of concern. In particular, the cognitive use of artifacts has been introduced by describing some meaningful examples. First, the twofold role played by artifacts once they are used by a cognitive agent is described. On the one side artifacts are supposed to provide operations, which agents can exploit to perform activities and attain their goals (purposive function). On the other side artifacts embeds information which is readable by agents to improve their epistemic states and can be considered as repositories of relevant information in working

environments (epistemic function). On these basis, a series of possible interaction styles are envisaged, either in providing relevant, strategic Information and easing decision making, either in changing means ends reasoning and in augmenting, through artifact operations, the capabilities to attain goals.

Second, the capability to "learn" and internalize artifact functionalities has been described. Artifacts, in this perspective, are supposed to realize external modules that agents can dynamically exploit as *external* services to enhance their action repertoire and – more generally – their capability to execute tasks. To this end, a particular component of artifacts, namely the manual, can be inspected by agents as an artifact meta-descriptor. Manuals provide, in a agent readable format, a set of operative instructions on how to exploit their functions. Once the functions of a consulted artifact have been internalized, agents can dynamically compose their plans since the new actions added to their repertoire. This would require, for agents, some additional abilities to bring about a dynamic action repertoire, i.e. by mean of an additional planner component allowing to build sequences of interleaved actions to achieve goals.

Many research lines have been followed in recent works at investigating specific aspects of cognitive interactions between agents and artifacts. Among others, [96] investigated the problem from an AI perspective, devising the particular intelligence skills needed by agents to cognitively interact with artifacts. Besides, the particular epistemic and pragmatic functions that artifacts may play for agents engaged in complex tasks requiring coordination and externalization of activities have been further investigated using *Jason* as agent architecture in [106, 105]. A fully goal directed approach to artifact functions has been introduced in [109] adopting and integrating *Jadex* BDI agents and using declarative goals that are mapped into artifact operations. The specialization of agent perceptive activities in artifact based environments has been investigated in [116], while a perception filtering mechanism, allowing agents to finalize belief updates on the basis of a subjective esteem of pragmatic relevance of percepts, has been introduced in [81]. Other works have been addressed at investigating goal oriented approach to Web Services through artifacts: in particular, an extension of CArtAgO, namely CArtAgO-WS, has been proposed as programming model for artifact based infrastructures to be exploited for the management of complex service oriented architectures [110, 111].

Although a BDI-like model will be still adopted for agents across the examples, using in particular *Jason* as the reference language, the rest of this work assumes a very basic notion of agency. Indeed, weak assumptions will be made on the particular architecture, model and language adopted by agents. It has to be

remarked that either A&A and then CArtAgO natively support different degree of *openness*. Workspaces are *open for agents*, namely heterogenous agents can join and quit dynamically to the system. On the other hand, *workspaces are open for artifacts*, that can be dynamically replaced, instantiated and disposed on the basis of the available types. Making weak or no assumptions on the computational model characterizing agents allows to emphasize and preserve the characteristic of *openness* for the whole environment—to which artifact based infrastructures are, finally, addressed.

This aspect will be clarified in the next Chapter 5, that – in order to provide unambiguous specification of MAS conceived in terms of agents, artifact and workspaces – provides a formal description of the A&A - CArtAgO model, abstracting away from implementation details. A different perspective, that will allow the functional specification of global dynamics inside the workspace, is then described in Chapter 6.

Chapter 5

Artifact Based Environments: a Formal Model of CArtAgO

To rigorously define the semantics of agent-artifact interactions and artifact computational behavior, this chapter describes a formal model of artifact-based environments. The chapter defines the entities involved in a MAS based on agents artifacts and workspaces in terms of their configurations. Then it focuses in particular on the dynamics occurring inside a single workspace, involving in particular interactions between agents and artifacts, and provides a description using operational semantics.

5.1 Formalising Artifact-Based Environments

The purpose of the formal model is twofold: first to provide a clear and rigorous semantics of artifacts computational behavior and of agent-artifact interaction model; then, to make the integration of CArtAgO with existing agent programming languages (and platforms) more suitable.

The formalisation abstracts from many details as found in the concrete implementation (CArtAgO technology in this case and related bridges to agent platforms): however, it includes all the essential aspects that we deem as important for environment programming in practice, in this case using an artifact-based model.

5.2 Structures

We first describe the structure of the states of the transition systems, which correspond to workspace configurations. In the following, we use symbols starting with an upper-case letter to denote sets.

5.2.1 Agent Configuration

Definition 1 *(Agent configuration)* An agent configuration $ag \in Ag$ is represented by a tuple:

$$\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr} \rangle$$

where ag_{id} is the agent unique identifier, ag_s is the agent internal state – here we abstract from its specific structure – ag_{Ev} is ordered set of events collected in current agent execution cycle (described in the next Section 5.3), and finally ag_{pr} defines the agent computational behavior—we refer to it as "agent program" even if it includes aspects that concern both the program and the architecture of the agent.

Events perceivable by agents ($ev \in ag_{Ev}$) are represented by tuples $\langle ev_t, ev_v \rangle$ with information about the *type* and the *value* of the event. When agent-artifact interactions are of concern, different kind of events ev may occur. The possible types of the events are reported in Table 5.1, and include action events – i.e. events about the completion or failure of an action – and events generated by artifacts – related to events signalled by artifact operation execution, or originated by changes and updates of observable properties.

Finally, the agent program is modeled as a function:

$$ag_{pr}(ag_s, ag_{Ev}) : (ag'_s, ac_{\perp})$$

which, given the current state of the agent ag_s and current event set ag_{Ev}, computes the new state of the agent ag'_{st} and the action to execute ac[1].

Remarks: For simplicity we here limit the set of events processable by agents to those events involving agent artifact interactions. A more complete agent model should include a wider set of events, as for instance events originating from agent internal processing (i.e. internal events related to practical reasoning and belief update in BDI-like agents) and events generated by message exchange (i.e. new message arrived). Examples of this approaches can be found in [13, 31]

[1] The value ac can be not defined ($\perp$), representing those situations in which no action is chosen.

Event type ev_t	Event values ev_v	Description
signal	$\langle ar_{id}, s_t, s_v \rangle$	Signal generated by an artifact – s_t is an identifier of the type of the signal and s_v is the value
action_completed	$\langle ac, ac_{fb} \rangle$	Action completed with success. *ac* is the action completed, ac_{fb} is the action feedback
action_failed	$\langle ac, ac_{fb} \rangle$	Action failure. *ac* is the action failed, ac_{fb} is a description of the failure
prop_updated	$\langle ar_{id}, p_n, p_v \rangle$	Observable property updated – p_n is the name of the property and p_v is new value
prop_new	$\langle ar_{id}, p_n, p_v \rangle$	A new artifact observable property is observed – p_n is the name of the property and p_v is the initial value
prop_nomore_obs	$\langle ar_{id}, p_n \rangle$	An artifact observable property is no more observed – p_n is the name of the property

Table 5.1: Events perceivable by an agent in CArtAgO environments

5.2.2 Artifact Configuration

Definition 2 *(Artifact configuration)* An artifact configuration $ar \in Ar$ is defined according to the following tuple:

$$\langle ar_{id}, ar_t, I, O, P, V \rangle$$

The configuration includes the main elements that characterize artifacts as environment abstraction:

- *Artifact identifier*: an unique artifact identifier represented by ar_{id}.
- *Artifact type*: Each artifact is defined on the basis of its type as it is specified in $ar_t \in Art$. ar_t contains artifact specifications, name of the type, artifact program and an initialization function. In particular, the type is represented by a tuple:

  $$\langle art_n, art_{init}, art_{pr}, man \rangle$$

  where art_n is the type name, *man* represents the artifact manual, here modeled as a simple literal, art_{init} and art_{pr} define the computational behavior of the artifacts that are instances of this type.

- *Interfaces*: The set I includes the artifact interfaces, listing the global set of operations that can be currently triggered by external entities upon the artifact. In particular, I is the general interface of triggerable operations, and it includes includes two distinct interfaces:

$$I = \langle Ui, Li \rangle$$

  where Ui is the *usage interface*, including operations that can be used by agents and Li is the *link interface*, i.e. a set of operations that can be triggered by other artifacts (linking artifacts). Elements $op \in I$ include both the name of the operation and actual value of the parameters, namely $op = \langle op_{name}, Params \rangle$.

- *Ongoing operations*: The set O includes, from time to time, those operation steps to be still executed inside the artifact, namely all the references to the next atomic steps of operations which are not completed yet. Each entry in O is represented by a tuple $op_{req} = \langle op_{id}, op, tag_{\perp} \rangle \in O$ including an identifier of the operation, the operation name and params ($op = \langle om_{name}, Params \rangle$), and a $tag_{\perp}$ (possibly not defined, i.e. $\perp$) specified by the agent that executed the operation to mark the events generated by the operation execution (this aspect will be clarified later on).

- *Observable state:* P is the set of the observable properties, represented by tuples $\langle p_n, p_v \rangle$ keeping track of the name and current value of the property.

- *Inner (non-observable) state:* V is the set of inner variables of the artifact, represented by tuples $\langle v_n, v_v \rangle$ including the name and current value of a state variable.

Artifact Type and Program

As said, an artifact type is represented by $ar_t = \langle art_n, art_{init}, art_{pr}, man \rangle$, where the former element art_n identifies the type name. This section provides more insights on the latter elements of an artifact type. The *initialisation* function art_{init} is given by:

$$art_{init}(Params) : (Ui_0, P_0, V_0, Li_0, OR_0)$$

which defines how the artifact is initialised when it is created. A list of parameters *Params* are needed to initialise an artifact: namely, $Ui_0, P_0, V_0, Li_0, OR_0$ represent

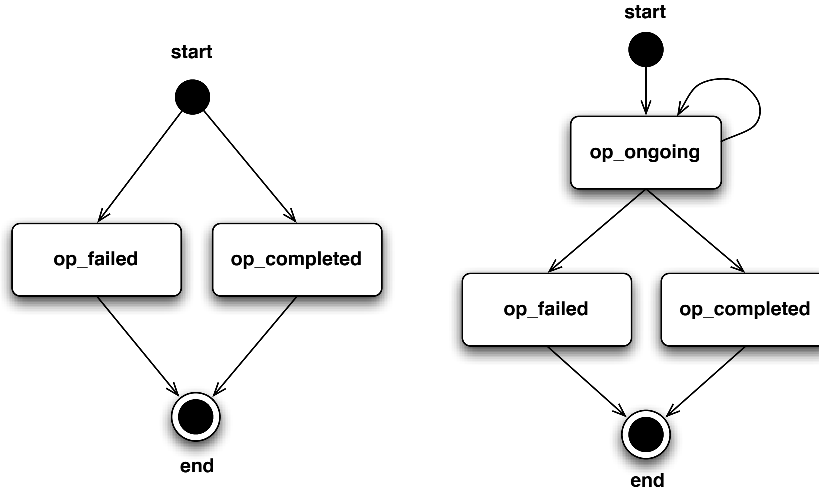

Figure 5.1: Operation State Transitions. An Operation can be in ongoing state if it is composed by a sequence of atomic operation steps.

respectively the initial value of the usage interface, observable property set, internal variable set, link interface, ongoing operation requests. art_{pr} is the artifact program, defining its functional specification. The program is represented by a partial function specifying the processes executed by the artifact during the execution of operation steps:

$$art_{pr}(op_{req}, I, P, V, t) : (Ui', P', V', Li', S, OR, LR, op_{cs}) \cup \bot$$

The parameters of an artifact program are operation request $op_{req} = \langle op_{id}, op, tag_{\bot} \rangle$ (each identified by an operation identifier op_{id}, operation name, parameters and tag) current interface $I = \langle Ui, Li \rangle$, current observable state P, current non-observable state V, and finally current workspace logic time t[2]. Given the input parameters, the function art_{pr} defines the value of the new usage interface Ui', observable state P', non-observable state V', the new link interface Li', the set of signals S possibly generated by the step execution. Observable properties in O are represented by tuples $\langle p_n, p_v \rangle$, including information about the type of the property and its value. Signals in S are represented by tuples $\langle s_t, s_v \rangle$, including information about the type of the signal and its value.

[2]The time is included among the parameters of the function in order to allow the definition of operation steps which are triggered at a specified time (an example is provided in the personal agenda artifact shown in Subsection 4.4.3, Figure 4.9).

Event type ev_t	Event value ev_v	Description
op_req	$\langle ag_{id}, ar_{id}, req_v, t\rangle$	An operation request is done by agent ag_{id} on artifact ar_{id} at time t – req_v identifies the value of the requested operation, namely op, that is operation name and params
stepop_req	$\langle ar_{id}, req_v, t\rangle$	A step operation request is added by the artifact ar_{id} to its queue of ongoing operations O at time t – req_v identifies the value of the requested operation step, namely op, that is operation step name and params

Table 5.2: Basic set of events related to operation requests collected by CArtAgO workspaces.

The function defines also the set of new operation requests OR specifying further operations to execute inside the artifact as a result of step execution and the set of link operation requests LR to trigger the execution of other operations in other artifacts. Elements in OR and LR are of the type $\langle req_t, req_v\rangle$, where $req_t \in \{\text{op_req}, \text{linkop_req}\}$ and req_v includes both the name of the requested operation and actual value of the parameters.

The value op_{cs} indicates the actual state of the operation (completion state). It is represented by a tuple $\langle op_s, op_{res}\rangle$, where:

- op_s is the operation state, which values $op_s \in \{\text{op_completed}, \text{op_ongoing}, \text{op_failed}\}$
- op_{res} is a value that can be specified to set the operation result (if any).

In turns, op_{cs} shows the possible state transitions for an operation (see also Figure 5.1). After a op_{req} (triggering event) the operation starts its execution and as soon as the operation completes, the state becomes op_completed. The state may become op_failed if any error occurs in the operation execution. An additional state is involved when the operation involves multiple steps. In this case the state becomes, after the execution of the first step, op_ongoing.

Event type ev_t	Event value ev_v	Description
focus_req	$\langle ag_{id}, req_v, t\rangle$	A focus request is done by Agent ag_{id} at time t – req_v identifies the focus action including the parameter ar_{id} as artifact to focus

stopFocus_req	$\langle ag_{id}, req_v, t\rangle$	A stopFocus request is done by Agent ag_{id} at time t – req_v identifies the stopFocus action including the parameter ar_{id} as artifact to stop focusing
obs_req	$\langle ag_{id}, ar_{id}, req_v, t\rangle$	An operation request is done by Agent ag_{id} on Artifact ar_{id} at time t – req_v identifies the the observeProp action including the parameter p_n as property name to observe
link_req unlink_req	$\langle ag_{id}, req_v, t\rangle$ $\langle ag_{id}, ar_{id}, req_v, t\rangle$	A request to link artifacts is done by Agent ag_{id} at time t – req_v identifies the linkArtifacts action, including the parameters (ar_{id}, ar'_{id}) as artifacts to be linked A request to unlink artifacts is done by Agent ag_{id} at time t – req_v identifies the unlinkArtifact action, including the parameters (ar_{id}, ar'_{id}) as artifacts to be unlinked
make_req dispose_req	$\langle ag_{id}, req_v, t\rangle$ $\langle ag_{id}, req_v, t\rangle$	A request to make a new artifact is done by Agent ag_{id} at time t – req_v identifies the makeArtifact action, including the initialisation parameters $art_n, Params$ for the artifact to be created A request to dispose an existing artifact is done by Agent ag_{id} at time t – req_v identifies the disposeArtifact action, including the identifier ar_{id} for the artifact to be disposed
lookup_req consult_req	$\langle ag_{id}, req_v, t\rangle$ $\langle ag_{id}, req_v, t\rangle$	A request to look for existing artifacts is done by Agent ag_{id} at time t – req_v identifies the lookupArtifact action, including the artifact type art_n for the artifacts to searched A request to consult the manual for an existing artifact type is done by Agent ag_{id} at time t – req_v identifies the consultMan action, including the artifact type art_n for the artifact to be consulted
join_req leave_req	$\langle ag_{id}, t\rangle$ $\langle ag_{id}, t\rangle$	A request to join the workspace is done by Agent ag_{id} at time t A request to leave the workspace is done by Agent ag_{id} at time t

Table 5.3: Additional events related to requests of agent activities collected by CArtAgO workspaces.

Event type ev_t	Event value ev_v	Description
op_signal	$\langle ar_{id}, s_t, s_v, t \rangle$	Signal generated by an artifact ar_{id} at time t – s_t is an identifier of the type of the signal and s_v is the value
op_ongoing	$\langle ar_{id}, ag_{id}, op_{req}, t \rangle$	Operation triggered by ag_{id}, described by $op_{req} = \langle op_{id}, op, tag_{\perp} \rangle$ finished a step in artifact ar_{id} at time t, but more steps are required to complete
op_completed	$\langle ar_{id}, ag_{id}, op_{req}, t \rangle$	Operation triggered by ag_{id}, described by $op_{req} = \langle op_{id}, op, tag_{\perp} \rangle$ completed with success in artifact ar_{id} at time t
op_failed	$\langle ar_{id}, ag_{id}, op_{req}, t \rangle$	Operation triggered by ag_{id}, described by $op_{req} = \langle op_{id}, op, tag_{\perp} \rangle$ failed in artifact ar_{id} at time t
prop_observed	$\langle ar_{id}, ag_{id}, p_n, p_v, t \rangle$	Observable property p_n has been observed in artifact ar_{id} by agent ag_{id} at time t – p_v is the actual property value
prop_updated	$\langle ar_{id}, p_n, p_v, t \rangle$	Observable property updated in artifact ar_{id} at time t – p_n is the name of the property and p_v is new value
prop_new	$\langle ar_{id}, p_n, p_v, t \rangle$	A new observable property is added in artifact ar_{id} at time t – p_n is the name of the property and p_v is the initial value
prop_removed	$\langle ar_{id}, p_n, t \rangle$	An observable property is removed in artifact ar_{id} at time t – p_n is the name of the property

Table 5.4: Basic set of events related to artifact changes collected by CArtAgO workspaces.

5.2.3 Workspace Configuration

Definition 3 *(Workspace configuration)* A workspace configuration is represented by the following tuple:

$$\langle Ag, Ar, Art, Ev, M, R, t \rangle$$

where:

Event type ev_t	Event value ev_v	Description
ar_created	$\langle ag_{id}, ar_{id}, ar_t, t\rangle$	An artifact ar_{id} is created by agent ag_{id} at time t, with ar_t indicating the type
ar_disposed	$\langle ag_{id}, ar_{id}, ar_t, t\rangle$	An artifact ar_{id} is disposed by agent ag_{id} at time t, with ar_t indicating the type
ar_looked	$\langle ag_{id}, art_n, t\rangle$	A look up action has been performed by agent ag_{id} for art_n at time t
ar_consulted	$\langle ag_{id}, ar_t, t\rangle$	A manual consult action has been performed by agent ag_{id} for ar_t – at time t
ar_focused	$\langle ag_{id}, ar_{id}, t\rangle$	Agent ag_{id} starts focusing Artifact ar_{id} at time t
ar_unfocused	$\langle ag_{id}, ar_{id}, t\rangle$	Agent ag_{id} stops focusing Artifact ar_{id} at time t
ar_linked	$\langle ar_{id}, ar'_{id}, t\rangle$	Artifact ar_{id}, ar'_{id} are linked at time t
ar_unlinked	$\langle ar_{id}, ar'_{id}, t\rangle$	Artifact ar_{id}, ar'_{id} are not linked at time t
ws_joined	$\langle ag_{id}, t\rangle$	An agent joined the workspace at time t, ag_{id} indicating agent identifier
ws_leaved	$\langle ag_{id}, t\rangle$	An agent leaved the workspace at time t, ag_{id} indicating agent identifier

Table 5.5: Basic set of events related to other agent activities collected by CArtAgO workspaces.

- Ag is a set of agents populating the workspace
- Ar is a set of artifacts actually created in the workspace
- Art is the set of artifact *types* actually available in the workspace
- Ev is a set collecting meaningful events launched inside the workspace. Generally speaking, each $ev \in Ev$ is represented by a tuple $\langle ev_t, ev_v\rangle$ with information about the *type* and the *value* of the event.
- M is a workspace map storing different kind of data which are need to govern global workspace dynamics. In particular the workspace map contains three elements $M = \langle Om, Lm, Um\rangle$, where each subset is intended at ruling over a particular aspect of agent-artifact interactions. In particular:
  - Om is an *observability* map, which elements $\in (Ag \times Ar)$ track who (agent) is observing what (artifact). Elements of the observability map

are tuples $\langle ag_{id}, ar_{id}, filter_f \rangle$, including the identifier of the observer agent, the identifier of the observed artifact, and a filtering function $filter_f$ (described in detail later on) that specifies the possible kinds of events that the observer agent is interested in.

- Lm is the *link* map with elements $\in (Ar \times Ar)$, tracking links between artifacts. Elements of the link map Lm are tuples $\langle ar_{id}, ar'_{id} \rangle$ containing the identifiers of the artifacts that are currently linked, being ar_{id} the source and ar'_{id} the target (that is, ar_{id} may trigger the execution of operations listed in the link interface of ar'_{id}).
- Um is the *usability* map indicating privileges of agents over artifact usage interfaces, i.e., if a specified agent has the *rights* to use specified artifact operations or not. By default every operation is usable by every agent, e.g. agents can use any declared operation included in any artifact Ui. Um bounds the usability privileges: its elements are represented by tuples like $\langle ar_{id}, ag_{id}, \{op_{name}\} \rangle$. Each Um entry tracks – for a given artifact ar_{id} – which are the *forbidden* operations ($\{op_{name}\}$) for a given agent ag_{id}.

The functioning and the mechanisms behind the two former map sets (Om, Lm) will be clarified later on this chapter, while the description of the policies operating upon the latter set (Um) will be described in Chapter 6.

- R is the set of workspace rules containing programmable rules aimed at correlating workspace events and at governing intra-workspace dynamics. Roughly speaking, workspace rules can be seen as environment programs, which execution is triggered once some specified events occur. A detailed description of workspace rules programming is outside the scope of this chapter, which is indeed intended at clarifying workspace dynamics and agent artifact interaction. A more complete description of workspace rules functioning and programming is provided in Chapter 6.

- t is a monotonically increasing time-stamp representing a logic notion time inside the workspace.

Given the above definition, different kind of events are possible inside the workspace as they are represented by general elements $ev \in Ev$, where each event is specified by its name and value: $ev = \{\langle ev_t, ev_v \rangle\}$.

- First, Ev includes events indicating the request pursued to trigger artifact operation, as the events triggering operation execution (a detailed description for this class of events is given in Table 5.2 and Table 5.3).

- Second, Ev includes events generated by artifacts and related to ongoing functioning of artifacts, as events signalled during operation execution or events originated by changes in observable properties (a detailed description for this class of events is given in Table 5.4).

- Third, Ev includes events related to activities performed by agents within the workspace, as agents entering and leaving the workspace, or creating/disposing, linking/unlinking artifacts etc. (a detailed description for this class of events is given in Table 5.5)

Remarks (a) We explicitly model the type of an artifact, which is what actually defines its computational behavior; so Ar can contain many instances on the same type $art \in Art$, each identified by a proper ar_{id}. **(b)** Being a single workspace not distributed, it is feasible to assume a unique notion of time shared among the agents and artifacts located in the same workspace. **(c)** In concrete implementations, agents can join a workspace even if they run on distributed platforms located in different nodes of the network. In order to situate agents in a given workspace, thus in order to provide to an agent the means to act and perceive, CArtAgO makes use of agent *bodies* [117]. More than a element of the programming model, an agent body has to be intended as part of the integration technology enabling agents to concretely operate in a workspace.

5.2.4 Workspace Initial Configuration

Definition 4 *(workspace initial configuration)* Let $\langle Ag, Ar, Art, Ev, M, R, t\rangle$ be the configuration of the workspace, where Ag is the set of agents, Ar is the set of artifacts, Art the set of artifacts types, Ev the set of events, M the set of workspace maps, R the set of workspace rule. Then, the initial configuration of a workspace is: $\langle \emptyset, \emptyset, Art, \emptyset, \emptyset, \emptyset, 0\rangle$, namely no agents nor artifacts are present in the workspace at time 0, while the event set is empty as well as the workspace maps and the set of workspace rules.

5.2.5 MAS Configuration

Definition 5 *(MAS configuration)* Being $\langle Ag, Ar, Art, Ev, M, R, t\rangle$ the configuration of a generic workspace, where the set of agents Ag is represented by elements like $\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle$, and the set of artifacts Ar is represented by elements like $\langle ar_{id}, ar_t, I, O, P, V\rangle$, then the configuration of the global multi agent system is given by elements in $\langle Ws\rangle$ represented by:

$$\langle ws_n, \quad \langle Ag, Ar, Art, Ev, M, R, t\rangle\rangle$$

and where ws_n represent an identifier for the specified workspace.

Remark In concrete implementation, being workspaces distributed across different nodes of the network, the workspace identifier also includes an address space.

5.3 Dynamics

Once the general structures of the entities populating a workspace have been provided, the main transition rules defining how the various configurations evolves will be described.

We model a workspace as a transition system $(W, \longrightarrow)$, where W is the set of states representing all the possible configurations that a workspace can assume, and $\longrightarrow \subseteq W \times W$ is a binary relation over W, describing how the workspace evolves from configuration to configuration, given the computational behavior of the agents and of the environment. We use the infix notation $w \longrightarrow w'$ to denote that $(w, w') \in \longrightarrow$.

Some conventions are used in the formal description. Sets are denoted by capital characters. Being S a set composed by elements $\langle X, Y, Z\rangle$, we denote S_X, S_Y and S_Z, to refer respectively to the X, Y and Z components of S. For instance, being the interface set of an artifact $I = \langle Ui, Li\rangle$, I_{Ui} and I_{Li} are used to indicate the usage interface Ui and the link interface Li in I, and being the workspace map $M = \langle Om, Lm, Um\rangle$, then M_{Om}, M_{Lm} and M_{Um} are used to indicate elements in Om, Lm and Um respectively.

To make the description more understandable, in the definition of the transition rules we include only those structures that are actually changed by the transition, sometimes splitting the description of the overall effect of a transition in multiple rules, in particular when the transition affects multiple structures of the workspace configuration. Notice that, for sake of simplicity, we omit the description of those transitions that are related to failures.

5.3.1 Agent Execution Cycle

Definition 6 *(agent execution cycle)* Being $\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle$ the configuration of an agent $ag \in Ag$, then:

$$\frac{ag_{pr}(ag_s, ag_{Ev}) = (ag'_s, ac_\perp)}{\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle \longrightarrow \langle ag_{id}, ag'_s, ag'_{Ev}, ag_{pr}\rangle}$$

that is, the configuration of an agent evolves as specified by the agent program, given the current state of the agent ag_s and and the set of events ag_{Ev} collected by the agent perceptive activities in current execution cycle. The transition leads to a new set of events ag'_{Ev}, whose value depends on the result of the execution of the action $ac_\perp$—this is detailed in next subsections. If no action chosen, or the skip the action is chosen, ac is $\perp$. In this case ag'_{Ev} is $\emptyset$.

Remarks: As noticed in Subsection 5.2.1, we here narrow the set of events processable by agents to those involving agent artifact interactions, thus not considering other type of events as the ones related to practical reasoning, belief update and message exchange.

5.3.2 Artifacts Dynamics

The computational dynamics of the environment are given by the processes executed inside artifacts. The execution of processes inside an artifact is due to operation execution. Artifact operations can be triggered in four different ways: *(a)* by the use action of agents; *(b)* by other operations in execution inside the same artifact; *(c)* by linking artifacts; *(d)* by workspace rules. In the following an operational semantic for the first three functioning is provided, while the model for workspace rules is described in Chapter 6

Agents using Artifacts

To act upon an artifact operation the action:

$$\mathsf{use}(ar_{id}, op, tag_\perp, align_f \cup \perp)$$

is provided to agents. The action triggers the execution of an operation $op = \langle op_{name}, Params\rangle$ on the target artifact ar_{id}, possibly specifying a symbolic *tag* to mark the events generated by the operation execution and an *alignment* condition function $align_f$. The tag can be exploited on the observation side to select only

those percepts that have been generated by a specific artifact, in the context of a specific operation execution. The alignment condition function $align_f : \mathsf{P} \rightarrow \{true, false\}$ can be specified by the agent programmer in order to define the condition on the observable state of the artifact that the agent expects to be true when the operation is actually triggered. Specifying $\perp$ in this case means an alignment function which is true for each value of the observable properties. The alignment condition function is necessary when there is the need to enforce consistency between the environment state expected by the agent acting upon the artifact and the actual state of the artifact when such an action takes place.

The execution of a use action elicits a sequence of transitions governed by specific events, each involving specific parts of the system. First, a op_req event indicating the request to use an artifact operation performed by an agent is registered by the workspace and it is added to the event set Ev. Formally:

Definition 7 *(use action)* Let $\langle Ag, Ar, Art, Ev, M, R, t\rangle$ be the configuration of the workspace, $\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle \in Ag$ the configuration of the agent doing a use action and $\langle ar_{id}, ar_t, I, P, V, O\rangle \in Ar$ the configuration of the target artifact. Being $ac = \mathsf{use}(ar_{id}, op, align_f, tag_\perp)$ the action selected by the agent, where $op = \langle op_{name}, Params\rangle$, then:

$$\frac{ev = \langle \mathsf{op_req}, \langle ag_{id}, ar_{id}, op, t\rangle\rangle}{\langle Ag, Ar, Art, Ev, M, R, t\rangle \longrightarrow \langle Ag, Ar, Art, Ev \cup ev, M, R, t'\rangle}$$

where the workspace event set Ev is updated with the op_req event.

After having received the request to serve an an artifact operation, the workspace addresses the request to the target artifact. Indeed, the presence of a op_req in the workspace set of events elicits the synchronous execution of an operation (or the first step of the operation, if it is a multi step operation). During the operation execution new events can be launched by the artifacts, that in short are further added in the workspace event set Ev. If the operation has a single step, then the artifact operation and the agent use action complete in a single transition and the set of workspace events Ev is updated to include the event about action completion. Otherwise, the operation (and the action) will be eventually completed by the execution of further operation steps. Formally:

Definition 8 *(use action execution)* Let $\langle Ag, Ar, Art, Ev, M, R, t\rangle$ be the configuration of the workspace, $\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle \in Ag$ the configuration of the agent doing a use action and $\langle ar_{id}, ar_t, I, P, V, O\rangle \in Ar$ the configuration of the target artifact. Being $\langle art_n, art_{init}, art_{pr}, man\rangle \in Art$ the type of the artifact, and op_{id} is a

fresh operation identifier in op_{req}, then:

$$\frac{ev = \langle \mathsf{op_req}, \langle ag_{id}, ar_{id}, op, t \rangle \rangle \quad ev \in Ev \quad \mathsf{triggering}(ev, R) = \bot \\ op \in I_{Ui} \quad \langle ag_{id}, ar_{id}, op \rangle \notin M_{Um} \quad align_f(P) = \mathsf{true} \\ art_{pr}(op_{req}, I, P, V, t) = (Ui', P', V', Li', S, OR, LR, op_{cs})}{\langle ar_{id}, ar_t, I, P, V, O \rangle \longrightarrow \langle ar_{id}, ar_t, I', P', V', O \cup O' \rangle}$$

The transition is triggered when an event ev is collected in the workspace set Ev, and only if this event is not triggering the execution of a workspace rule in R, as specified by the function $\mathsf{triggering}(ev, R)$[3]. Moreover, the transition is triggered only if the requested operation is part of the artifact usage interface I_{Ui}, if the agent has the rights to use the operation op according to the usability map Um, and if the align function is satisfied given the actual artifact observable state P. Notice that if the operation is multi-step, it does not complete in one transition, thus the set of ongoing operations O is updated to O'. In this case a new element $op_{req} = \langle op_{id}, op, tag_{\bot} \rangle$ is added to the artifact, representing the operation itself to be completed A new element $op'_{req} = \langle op'_{id}, op', \bot \rangle$ is further added to O for each $op'_{req} \in OR$, that is for each new operation step triggered by the artifact itself. Finally, the transition may also affect the artifact interface I and the artifact internal variables V according to the particular specification provided by the artifact program art_{pr}.

As an artifact has executed an operation step, the workspace configuration changes accordingly:

$$\frac{ev = \langle \mathsf{op_req}, \langle ag_{id}, ar_{id}, op, t \rangle \rangle \quad \mathsf{triggering}(ev, R) = \bot \\ op \in I_{Ui} \quad \langle ag_{id}, ar_{id}, op \rangle \notin M_{Um} \quad align_f(P) = \mathsf{true} \\ art_{pr}(op_{req}, I, P, V, t) = (Ui', P', V', Li', S, OR, LR, op_{cs})}{\langle Ag, Ar, Art, Ev \cup ev, M, R, t \rangle \longrightarrow \langle Ag, Ar, Art, Ev \cup ev', M, R, t' \rangle}$$

where the new set of events is updated with the overall set of events originated by the operation execution. In this case, the added events are denoted by ev'. They can be new events related to linking requests (as detailed in Table 5.2) or to artifact changes (as detailed in Table 5.4). In particular:

- A new event $ev' = (\mathsf{linkop_req}, \langle ar_{id}, ar'_{id}, req_v, t \rangle)$ is added for each linking request $req_v \in LR$ addressed to another artifact ar'_{id}. The event is added only

[3]The function $\mathsf{triggering}(ev, R)$ will be specified in Chapter 6. For the moment it is enough to consider that this function returns $\bot$ when the set R is not containing a rule entry indicating ev as its triggering condition.

if the two artifacts are linked according to the linkability map, namely only if $\langle ar_{id}, ar'_{id} \rangle \in Lm$.

- A new event $ev' = (\mathsf{op_signal}, \langle ar_{id}, s_t, s_v, t \rangle)$ is added for each signal generated by the operation execution as detailed by S, being s_t the identifier of the type of the signal and s_v its value.
- A new event $ev' = (\mathsf{op_completed}, \langle ar_{id}, ag_{id}, op_{req}, t \rangle)$ is generated if the operation completed with success, as detailed in op_{cs}.
- A new event $ev' = (\mathsf{op_ongoing}, \langle ar_{id}, ag_{id}, op_{req}, t \rangle)$ is generated if the operation finished a step but more steps are required to complete, as detailed in op_{cs}.
- A new event $ev' = (\mathsf{op_failed}, \langle ar_{id}, ag_{id}, op_{req}, t \rangle)$ is generated if the operation failed, as detailed in op_{cs}.
- A new event $ev' = (\mathsf{prop_updated}, \langle ar_{id}, p_n, p_v, t \rangle)$ is added for each observable property updated – as detailed in P', being p_n the name of the property and p_v its updated value.
- A new event $ev' = (\mathsf{prop_new}, \langle ar_{id}, p_n, p_v, t \rangle)$ is added for each observable property created – as detailed in P', being p_n the name of the property and p_v its new value.
- A new event $ev' = (\mathsf{prop_removed}, \langle ar_{id}, p_n, p_v, t \rangle)$ is added for each observable property removed – as detailed in P', being p_n the name of the property and p_v its value.

Finally, all the agents that are perceiving the artifact receive the events related to operation execution. For agents, the semantics of action execution is that the action completes (fails) when the requested operation completes (fails). We refer to action feedback as the result of the operation execution. Besides action feedback/result, independently from the completion or failure of the operation, during an action execution the artifact may generate observable events that the agent can perceive. Those events can be elicited since a change in artifact observable properties, or since a signal generated by the artifact during its operation execution. This kind of events, in particular, can be marked by a specified tag, by which the agent can filter their values. In this case the model differs between agents which are focusing the artifact executing the operation step and the agent that triggered

the execution of the operation. In the first case, a set of transitions updates the event set ag_{Ev} for all the agents actually focusing the artifact according to the observability map *Om*. In the second case, the agent is further acknowledged with a percept indicating the action feedback in terms of operation completion state. Formally:

Definition 9 *(use action on agents)* Let $\langle Ag, Ar, Art, Ev, M, R, t\rangle$ be the configuration of the workspace, $\langle ar_{id}, ar_t, I, P, V, O\rangle \in Ar$ the configuration of the artifact that executes the operation step and $\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle \in Ag$ the configuration of one agent focusing it. Then, for each event *ev* produced during the operation execution, the agent focusing the artifact is affected by the following transition:

$$\frac{ev \in Ev \quad \langle ag_{id}, ar_{id}, tag\rangle \in Om \qquad art_{pr}(op_{req}, I, P, V, t) = (Ui', P', V', Li', S, OR, LR, op_{cs})}{\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle \longrightarrow \langle ag_{id}, ag_s, ag_{Ev} \cup ev, ag_{pr}\rangle}$$

where the added event *ev* updating the agent configuration can include both events generated by changes in the observable properties of the artifact (as specified in P') or signals generated by the artifact in its operation step (as included in$S$). In particular:

- In the first case, according to the new artifact configuration P', *ev* may be a prop_updated event (if some observable property has been updated during artifact operation), but also prop_new and prop_nomore_obs (if properties have been added or removed).

- In the second case *ev* can be related to a signal events, specified by type/name pairs ($\langle s_t, s_v\rangle$).

The complete description of these events is given in Table 5.1. Notice that to be added to the observability map and to specify filters, an agent has to execute a focus action, which is described later on in this section. Otherwise, if the agent is not observing the artifact according to the observability map *Om*, then ev' is null. Moreover, all the events are filtered by the filter function $filter_f$ possibly specified in the observability map. Given this, the set contains only those events ev' such that $filter_f(ev')$ is true.

Besides the previous updates, (only) the agent that triggered the artifact operation is affected by the following transition:

$$\frac{ev \in Ev \quad ev_t \in \{\mathsf{op_completed}, \mathsf{op_failed}, \mathsf{op_ongoing}\} \quad ag_{id} \in ev_v \qquad art_{pr}(op_{req}, I, P, V, t) = (Ui', P', V', Li', S, OR, LR, op_{cs})}{\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle \longrightarrow \langle ag_{id}, ag_s, ag_{Ev} \cup ev', ag_{pr}\rangle}$$

where ev' can be related to the operation completion state op_{cs}, namely action_completed, action_failed if the operation completed with success/failure. In this case a translation is done from the event ev as it is produced by ar_{id} to the event ev', as it is received by ag_{id}. In particular:

- If $ev = \langle \mathsf{op_completed}, \langle ar_{id}, ag_{id}, op_{req}, t \rangle \rangle$, then $ev' = \langle \mathsf{action_completed}, \langle ac, ac_{fb} \rangle \rangle$.
- If $ev = \langle \mathsf{op_failed}, \langle ar_{id}, ag_{id}, op_{req}, t \rangle \rangle$ then $ev' = \langle \mathsf{action_failed}, \langle ac, ac_{fb} \rangle \rangle$.
- If $ev = \langle \mathsf{op_ongoing}, \langle ar_{id}, ag_{id}, op_{req}, t \rangle \rangle$ then $ev' = \emptyset$.

where the value of the action feedback (ac_{fb}) is set on the basis of the operation completion state op_{cs}, as it is given by the artifact program (art_{pr}).

Remarks: (a) The time t is automatically changed by the underlying execution system, using a discrete, linear notion of time. Henceforth , for the purpose of the operational semantics, it is assumed that all rules that apply at a given time are actually applied before the system changes the state to the next time. **(b)** The function $\mathsf{triggering}(ev, R)$ is defined locally to the workspace, and it provides $\perp$ if the event ev is not matching with some rule defined in the set R, otherwise it returns the rule which head has unified with ev. The matching mechanisms also involves some further conditions to be checked in the overall workspace states. The mechanisms for the application of the rules in R, as well as the involved transitions related to R, will be provided in Chapter 6 together with a formal description of the function triggering. **(c)** In concrete implementation the translation from the event ev as it is produced by ar_{id} to the event ev', as it is received by ag_{id} is done by the particular integration technology, that is the bridge mechanism integrating the particular agent architecture to CArtAgO.

Multi step Operations

Multi step operations can be triggered by agent's use actions. A multi step operation elicits the execution of a sequence of atomic operation steps inside the artifact. Whereas the first step of the sequence is executed according to the rules described above, the following steps are executed as soon as the artifact set O – which includes ongoing operation requests to be served – is not empty and when the artifact program art_{pr} for that request – given the current state of the artifact – yields to a new valid configuration $\neq \perp$. Formally:

Definition 10 *(operation step)* Being $\langle Ag, Ar, Art, Ev, M, R, t \rangle$ the workspace configuration, $\langle ar_{id}, ar_t, I, P, V, O \rangle \in Ar$ an artifact configuration and being the artifact

type $\langle art_n, art_{init}, art_{pr}, man \rangle \in Art$, then:

$$\frac{\langle op_{req}, tag_\perp \rangle \in O \qquad art_{pr}(op_{req}, I, P, V, t) = (Ui', P', V', Li', S, OR, LR, op_{cs}) \neq \perp}{\langle ar_{id}, ar_t, I, P, V, O \cup \langle op_{req}, tag_\perp \rangle \rangle \longrightarrow \langle ar_{id}, ar_t, I', P', V', O' \rangle}$$

where O' includes the originating operation $\langle op_{req}, tag_\perp \rangle$ only if the operation, by executing the current step, has not completed (according to the value of op_{cs}). O' includes also a new element $\langle op'_{req}, \perp \rangle$, for each $op'_{req} \in OR$, i.e. for each new operation triggered by the execution of the step (where $op'_{id} \in op'_{req}$ is a fresh operation identifier). As in the case of single step operations, the execution of each step of a multi step operation may affect the artifact interface I and the artifact internal variables V according to the particular program specified by art_{pr}.

Accordingly, the workspace configuration changes as it follows:

$$\frac{\langle op_{req}, tag_\perp \rangle \in O \qquad art_{pr}(op_{req}, I, P, V, t) = (Ui', P', V', Li', S, OR, LR, op_{cs}) \neq \perp}{\langle Ag, Ar, Art, Ev \cup ev, M, R, t \rangle \longrightarrow \langle Ag, Ar, Art, Ev', M, R, t \rangle}$$

where the new set of events Ev' is updated with the overall set of events originated by the step execution. As in the *use action*, $Ev' = Ev \cup Ev_{new}$, where Ev_{new} can be new events related to linking requests (as detailed in Table 5.2) or to artifact changes (as detailed in Table 5.4).

On the agent side, the transition given by the step execution causes the update of the event set ag_{Ev} for each agent $ag_{id} \in Ag$ observing the artifact according to the observability map Om. In this case the new set ag'_{Ev} includes the events generated by the step execution. For simplicity, being these transitions analogous to the ones defined in the *use action*, a detailed description is omitted.

Remarks: **(a)** The same mechanism and the same rules for executing operation steps are applied both to multi-step operations and to artifact internal operations. **(b)** No operation feedback is received in terms of op_{cs} by the agent who has initiated the sequence of a multi step operation. Nevertheless the agent can still monitor ongoing operation by focusing the artifact, as described in the following sections.

Artifact linked Operations

Link operations are executed inside artifacts in order to serve operations triggered by other artifacts. The event initiating the execution of a link operation is registered in the workspace set Ev. In particular, once the workspace event set contains

an event of the type linkop_req, and if the linking artifact is registered in the linkability map *Lm*, the request is addressed to the linked artifact in order to serve the ink request. Formally:

Definition 11 *(linkop execution)* Let $\langle Ag, Ar, Art, Ev, M, R, t\rangle$ be the workspace configuration , $\langle ar_{id}, ar_t, I, P, V, O\rangle \in Ar$ the configuration of a linked artifact and $\langle ar'_{id}, ar'_t, I', P', V', O'\rangle \in Ar$ the configuration of a linking artifact. Let be art_{pr} is the linked artifact program, as defined by its type $\langle art_n, art_{init}, art_{pr}, man\rangle \in Art$. Being $ev = \langle \mathsf{linkop_req}, \langle ar'_{id}, ar_{id}, req_v, t\rangle\rangle$ a link operation request from ar'_{id} to ar_{id} and being $ev \in Ev$, then the transition affecting the workspace is the following:

$$\frac{ev \in Ev \quad \mathsf{triggering}(ev, R) = \bot \quad \langle ar'_{id}, ar_{id}\rangle \in M_{Lm} \quad op \in I_{Li} \\ art_{pr}(op_{req}, I, P, V, t) = (Ui', P', V', Li', S', OR, LR', op_{cs}) \neq \bot}{\langle Ag, Ar, Art, Ev \cup ev, M, R, t\rangle \longrightarrow \langle Ag, Ar, Art, Ev', M, R, t'\rangle}$$

where the new set of events Ev' is updated with the overall set of events originated by operation execution. as in the *use action*, $Ev' = Ev \cup Ev_{new}$, where Ev_{new} can be new events related to new linking requests (as detailed in Table 5.2) or to artifact changes (as detailed in Table 5.4).

Accordingly, the transition involving the linked artifact is the following:

$$\frac{ev \in Ev \quad \mathsf{triggering}(ev, R) = \bot \quad \langle ar'_{id}, ar_{id}\rangle \in Lm \quad op \in I_{Li} \\ art_{pr}(op_{req}, I, P, V, t) = (Ui', P', V', Li', S', OR, LR', op_{cs}) \neq \bot}{\langle ar_{id}, ar_t, I, P, V, O\rangle \longrightarrow \langle ar_{id}, ar_t, I', P', V', O'\rangle}$$

where I', P', V', O' undergo the changes elicited by the link operation step execution and are computed as shown in previous transition rules.

Finally, the transition $ag_{Ev} \longrightarrow ag'_{Ev}$ affects agents ag_{id} observing the artifact according to the observability map *Om*. Also this transition follows the rules defined above (and is not further detailed here).

Remark: For simplicity we assumed that is not feasible to specify a link operation as a multi step operation. Given this, triggering steps inside a link operation is forbidden. Otherwise, a link operation can trigger further link requests according to the local set *LR*, as it can be modified by the artifact program.

5.3.3 Agent Perceptive Activities

Agent perceptive activities concerns mechanisms enabling agents to update their knowledge by inspecting the environment. Once artifact based environments are

of concern, the basic building block inspectable by agent is the artifact. As said, an artifact has two basic ways to be inspected, namely its set of observable properties P and all the events generated since an operation execution. Hence, agents have two approaches to inspect artifacts, namely to observe their properties in P, or to process (and possibly filter) noticeable events elicited by operation execution. On these basis, agent perceptive activities are conceived on the basis of two different actions:

Focusing agent ability to *select* which parts of the environment to perceive, namely focusing only on specific subset of artifacts;

Observation agent ability to synchronously read the value of an observable property of an artifact.

Focusing Artifacts

The first capability is provided by the focus action:

$$\mathsf{focus}(ar_{id}, filter_f)$$

where ar_{id} is the identifier of the artifact to observe and $filter_f$ is a boolean function $filter_f : \mathsf{Ev} \longrightarrow \{true, false\}$ representing a filtering condition for events specifying that an event is collected only if the filter on the event is true. Formally:

Definition 12 *(focus action)* Let be $\langle Ag, Ar, Art, Ev, M, R, t\rangle$ the configuration of the workspace, $\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle \in Ag$, the agent configuration and let be $\langle ar_{id}, ar_t, I, P, V, O\rangle \in Ar$ the artifact configuration. Being the action to be performed by agent $ac = \mathsf{focus}(ar_{id}, filter_f)$, and being $ev = \langle \mathsf{focus_req}, \langle ag_{id}, req_v, t\rangle\rangle$ then the action succeeds and:

$$\frac{ev \in Ev \quad \mathsf{triggering}(ev, R) = \bot}{\langle Ag, Ar, Art, Ev \cup ev, M, R, t\rangle \longrightarrow \langle Ag, Ar, Art, Ev', M', R, t'\rangle}$$

where:

- M_{Om} is updated in M', in particular $Om' = Om \cup \{\langle ag_{id}, ar_{id}, filter_f\rangle\}$, the observability map is updated by adding the information about which agent is observing which artifact, with the function $filter_f$
- $Ev' = Ev \cup \langle \mathsf{ar_focused}, \langle ag_{id}, ar_{id}, t\rangle\rangle$, indicating that agent ag_{id} has focused artifact ar_{id} at time t

On the agent side $ag'_{Ev} = \{\langle \mathsf{action_completed}, \langle \mathsf{focus}(ar_{id}, filter_f), \bot \rangle \rangle\} \cup Ev_{prop}$, where Ev_{prop} is a set of events of type prop_new, one for each observable property $\langle p_n, p_v \rangle \in P$.

Dually to focus, an action is provided to agents to stop getting events related to a specific artifact:

$$\mathsf{stopFocus}(ar_{id})$$

This is modeled by a transition rule removing the element from the observability map and adding the related events to the workspace set Ev. Formally:

Definition 13 *(stop-focus action)* Let $\langle Ag, Ar, Art, Ev, M, R, t \rangle$ be the configuration of the workspace, $\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr} \rangle \in Ag$, the agent configuration and $\langle ar_{id}, ar_t, I, P, V, O \rangle \in Ar$ the artifact configuration. Being the $ac = \mathsf{stopFocus}(ar_{id})$ the action to be performed by the agent, and being $ev = \langle \mathsf{stopFocus_req}, \langle ag_{id}, req_v, t \rangle \rangle$ then the action succeeds and:

$$\frac{ev \in Ev \quad \mathsf{triggering}(ev, R) = \bot}{\langle Ag, Ar, Art, Ev \cup ev, M, R, t \rangle \longrightarrow \langle Ag, Ar, Art, Ev', M', R, t' }$$

where:

- M_{Om} is updated in M', in particular $Om' = Om \setminus \{\langle ag_{id}, ar_{id}, filter_f \rangle\}$, namely the observability map is updated by removing the entry about the couple ag_{id}, ar_{id}.
- $Ev' = Ev \cup \langle \mathsf{ar_unfocused}, \langle ag_{id}, ar_{id}, t \rangle \rangle$, indicating that agent ag_{id} has finished to focus artifact ar_{id} at time t.

On the agent side the set of events is updated with the action feedback, related to the action completion, and with the events related to the properties which are no more observable. In more detail, $ag'_{Ev} = \{\langle \mathsf{action_completed}, \langle \mathsf{stopFocus}(ar_{id}), \bot \rangle \rangle\} \cup Ev_{prop}$, where Ev_{prop} is a set of events ev of type prop_nomore_obs, one for each observable property $\langle p_n, p_v \rangle \in P$.

Observing Artifact Properties

The capability to synchronously read the value of an observable property is provided by the observeProp action:

$$\mathsf{observeProp}(ar_{id}, p_n)$$

The action acts on the artifact ar_{id} and retrieves the value of the observable property p_n as action feedback. Formally:

Definition 14 *(observe property action)* Let $\langle Ag, Ar, Art, Ev, M, R, t\rangle$ be the configuration of the workspace, $\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle \in Ag$ the agent configuration and $\langle ar_{id}, ar_t, I, P, V, O\rangle \in Ar$ the artifact configuration. Being $\mathsf{observeProp}(ar_{id}, p_n)$ the action to do such that $\langle p_n, p_v\rangle \in P$, and being $ev = \langle \mathsf{obs_req}, \langle ag_{id}, ar_{id}, req_v, t\rangle\rangle$ then the action completes in the same transition:

$$\frac{ev \in Ev \quad \mathsf{triggering}(ev, R) = \bot}{\langle Ag, Ar, Art, Ev \cup ev, M, R, t\rangle \longrightarrow \langle Ag, Ar, Art, Ev', M, R, t'}$$

where Ev is updated with the event $ev = \langle \mathsf{prop_observed}, \langle ar_{id}, ag_{id}, p_n, p_v, t\rangle\rangle$ indicating that artifact ar_{id}'s observable property p_n has been observed by agent ag_{id} at time t – p_v is the observed property value.
On the agent side, the transition is given by:

$$\frac{ev \in Ev \quad \mathsf{triggering}(ev, R) = \bot}{\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle \longrightarrow \langle ag_{id}, ag_s, ag'_{Ev}, ag_{pr}\rangle}$$

where ag_{Ev} is updated with the action feedback given by the value of the observable property p_v: $ag'_{Ev} = \{\langle \mathsf{action_completed}, \langle \mathsf{observeProp}(ar_{id}, ar_n), p_v\rangle\rangle\}$.

Remark: Actions related to environment observation do not affect artifact configurations. observeProp action in particular does not change any structure of the artifact configuration.

5.3.4 Agents Joining and Leaving Workspaces

Once multi-agent system including multiple workspaces are of concern, agents are allowed to dynamically join and quit a workspace, possibly working in multiple workspaces at a time. Two actions are provided to agents in order to enter and leave workspaces. To enter in a workspace identified by ws_n (workspace name) agents can execute the action:

$$\mathsf{joinWorkspace}(ws_n)$$

that is, by executing the action the agent receives, in case of success, a unique agent identifier ag_{id} assigned by the workspace.

Definition 15 *(join workspace)* Let $\langle Ag, Ar, Art, Ev, M, R, t\rangle$ be the configuration of the workspace identified by ws_n and $\langle _, ag_s, ag_{Ev}, ag_{pr}\rangle$ the configuration of an

agent $\notin Ag$, thus with an undefined identifier. Being $ac = \mathsf{joinWorkspace}(ws_n)$ the action executed by the agent to enter in a workspace and being $ev = \langle \mathsf{join_req}, \langle ag_{id}, t \rangle \rangle$ the event indicating the request done by the agent to join the workspace at time t, then, in case of success the workspace configuration changes as it follows:

$$\frac{ev \in Ev \quad \mathsf{triggering}(ev,R) = \bot \quad ag_{id} \notin Ag}{\langle Ag, Ar, Art, Ev \cup ev, M, R, t \rangle \longrightarrow \langle Ag', Ar, Art, Ev', M, R, t' \rangle}$$

where:

- The updated set Ag is updated including the joining agent: $Ag' = Ag \cup ag_{id}$
- The updated set Ev includes the event related to the joined agent: $Ev' = \{\mathsf{ws_joined}, \langle ag_{id}, t' \rangle\}$, where ag_{id} is a fresh identifier.

On the agent side the following transition can be defined:

$$\frac{ev \in Ev \quad \mathsf{triggering}(ev,R) = \bot}{\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr} \rangle \longrightarrow \langle ag_{id}, ag_s, ag'_{Ev}, ag_{pr} \rangle}$$

where: $ag'_{Ev} = \{\mathsf{action_completed}, \langle \mathsf{joinWorkspace}, ag_{id} \rangle\}$. In this case the action feedback is the fresh identifier ag_{id} assigned to the agent.

Dually to joinWorkspace, the $\mathsf{quitWorkspace}(ws_n)$ is provided to leave a workspace. This is modeled by a transition rule removing the agent from the workspace Ag set.

Definition 16 *(quit workspace)* Let $\langle Ag, Ar, Art, Ev, M, R, t \rangle$ be the configuration of the workspace identified by ws_n, $\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr} \rangle$ the configuration of the agent. Being $ac = \mathsf{quitWorkspace}(ws_n)$ the action to leave a workspace and being $ev = \langle \mathsf{leave_req}, \langle ag_{id}, t \rangle \rangle$ an event indicating the request done by Agent ag_{id} to leave the workspace at time t then, in case of success the workspace configuration changes as follows:

$$\frac{ev \in Ev \quad \mathsf{triggering}(ev,R) = \bot \quad ag_{id} \in Ag}{\langle Ag, Ar, Art, Ev \cup ev, M, R, t \rangle \longrightarrow \langle Ag', Ar, Art, Ev', M, R, t'}$$

where:

- The updated set Ag is updated including the joining agent: $Ag' = Ag \setminus ag_{id}$
- The updated set Ev includes the event related to the joined agent: $Ev' = \{\mathsf{ws_leaved}, \langle ag_{id}, t' \rangle\}$

On the agent side, the following transition can be defined:

$$\frac{ev \in Ev \quad \mathsf{triggering}(ev,R) = \bot \quad ag_{id} \in Ag}{\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle \longrightarrow \langle ag_{id}, ag_s, ag'_{Ev}, ag_{pr}\rangle}$$

where: $ag'_{Ev} = \{\mathsf{action_completed}, \langle \mathsf{quitWorkspace}, \bot\rangle\}$ and no symbolic action feedback is provided.

5.3.5 Environment Management and Inspection

A set of actions are provided to agents for changing and exploring the structure of the environment, for instance by dynamically creating new instances of artifacts, disposing existing ones, linking and unlinking artifacts, looking up the presence of some type of artifacts and inspecting the manual of a specific artifact type. A formal description for these actions is provided in the following sections.

Creating and Removing Artifacts

In order to dynamically create a new artifact the action

$$\mathsf{makeArtifact}(ar_{id}, ar_t, Params)$$

is provided. It is assumed to create and initialise a new artifact ar_{id} since an artifact type ar_t and given the list of initializing parameters *Params*. Formally:

Definition 17 *(artifact creation)* Let $\langle Ag, Ar, Art, Ev, M, R, t\rangle$ be the configuration of the workspace and $\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle \in Ag$ the configuration of the agent. Being $ac = \mathsf{makeArtifact}(ar_{id}, ar_t, Params)$ the action to do, being ar_{id} a fresh identifier for the artifact to be created – which must be unique in the workspace – ar_t the type, and *Params* a list of parameters, and being $ev = \langle \mathsf{make_req}, \langle ag_{id}, req_v, t\rangle\rangle$, then:

$$\frac{\begin{array}{c} ev \in Ev \quad \mathsf{triggering}(ev,R) = \bot \\ \langle art_n, art_{init}, art_{pr}, man\rangle \in Art \quad art_{init}(Params) = (Ui_0, P_0, V_0, Li_0, OR_0) \end{array}}{\langle Ag, Ar, Art, Ev \cup ev, M, R, t\rangle \longrightarrow \langle Ag, Ar', Art, Ev', M, R, t'\rangle}$$

where:

- $Ar' = Ar \cup \{\langle ar_{id}, ar_t, Ui_0, P_0, V_0, Li_0, OR_0, \emptyset\rangle\}$ indicating the new set of artifacts actually dwelling the workspace.

- $Ev' = Ev \cup (\mathsf{ar_created}, \langle ag_{id}, ar_{id}, art, t \rangle)$ where the added event indicates that a new artifact ar_{id} has been created by ag_{id} at time t, with type art.

On the agent side: $ag'_{Ev} = \{\langle \mathsf{action_completed}, \langle \mathsf{makeArtifact}(ar_{id}, ar_t, Params), \bot \rangle \rangle\}$. In this case the action feedback is $\bot$ being the fresh identifier ar_{id} a logic name chosen by the agent and used as an input parameter for the action.

Remarks: **(a):** The makeArtifact in CArtAgO implementation provides as an output parameter the (system) identifier of a created artifact given its logic name. This is not included in the abstract model since we assume ar_{id} is a unique identifier, used for both the logic name and the system identifier.

(b): Agents can create only instances of artifacts whose type is available in the workspace, and different workspaces can have different set of types, even different types with the same name.

Artifact disposal is realized by means of a the action:

$$\mathsf{disposeArtifact}(ar_{id})$$

that is assumed to remove an existing artifact identified by ar_{id} in the workspace. Formally:

Definition 18 *(artifact disposal)* Let $\langle Ag, Ar, Art, Ev, M, R, t \rangle$ be the configuration of the workspace and $\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr} \rangle \in Ag$ the configuration of the agent. Being $ac = \mathsf{disposeArtifact}(ar_{id})$ the action to do, where ar_{id} the identifier of the artifact to be removed and being $ev = \langle \mathsf{dispose_req}, \langle ag_{id}, req_v, t \rangle \rangle$ then:

$$\frac{ev \in Ev \quad \mathsf{triggering}(ev, R) = \bot \quad ar_{id} \in Ar}{\langle Ag, Ar, Art, Ev \cup ev, M, R, t \rangle \longrightarrow \langle Ag, Ar', Art, Ev', M, R, t' \rangle}$$

where:

- $Ar' = Ar \setminus \{\langle ar_{id}, ar_t, I, P, V, O \rangle\}$ indicating the new set of artifacts actually dwelling the workspace.

- $Ev' = Ev \cup \langle \mathsf{ar_disposed}, \langle ag_{id}, ar_{id}, art, t \rangle \rangle$ where the added event indicates that an existing artifact ar_{id} with type art has been disposed by ag_{id} at time t.

On the agent side: $ag'_{Ev} = \{\langle \mathsf{action_completed}, \langle \mathsf{disposeArtifact}(ar_{id}), \bot \rangle \rangle\}$. In this case the action feedback is empty.

Linking and Unlinking Artifacts

Two actions are provided to link and unlinking artifacts. To link two artifacts the action

$$\mathsf{linkArtifacts}(ar_{id}, ar'_{id})$$

is provided, where $ar_{id} \in Ar$ is the identifier of the linking artifact (the source) and $ar'_{id} \in Ar$ is the identifier of the linked artifact (the target). If the action succeeds, the linking artifact can trigger the execution of operations on the linked artifact. Formally:

Definition 19 *(link artifacts action)* Let $\langle Ag, Ar, Art, Ev, M, R, t\rangle$ be the configuration of the workspace, $\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle \in Ag$ the configuration of the agent, and ar_{id}, ar'_{id} two artifact identifiers in Ar, with $ar_{id} \neq ar'_{id}$. Being $ac = \mathsf{linkArtifacts}(ar_{id}, ar'_{id})$ the action to do such that ar_{id} and ar'_{id} identify two not-linked artifacts of the workspace, and being $ev = \langle \mathsf{link_req}, \langle ag_{id}, req_v, t\rangle\rangle$ then:

$$\frac{ev \in Ev \quad \mathsf{triggering}(ev, R) = \bot \langle ar_{id}, ar'_{id}\rangle \notin Lm}{\langle Ag, Ar, Art, Ev \cup ev, M, R, t\rangle \longrightarrow \langle Ag, Ar, Art, Ev', M', R, t'\rangle}$$

where:

- $Ev' = Ev \cup \langle \mathsf{ar_linked}, \langle ar_{id}, ar'_{id}, t\rangle\rangle$
- M' contains an updated linkability map, $Lm' = Lm \cup \{\langle ar_{id}, ar'_{id}\rangle\}$

On the agent side $Ev' = \{\langle \mathsf{action_completed}, \langle \mathsf{linkArtifacts}(ar_{id}, ar'_{id}), \bot\rangle\rangle\}$.

Dually to linkArtifacts, the $\mathsf{unlinkArtifacts}(ar_{id}, ar'_{id})$ action unlinks two artifacts previously linked. This is modeled by a transition rule removing the element from the link map:

Definition 20 *(unlink artifact action)* Let $\langle Ag, Ar, Art, Ev, M, R, t\rangle$ be the configuration of the workspace, $\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle \in Ag$ the configuration of the agent, and ar_{id}, ar'_{id} two artifact identifiers in Ar, with $ar_{id} \neq ar'_{id}$. Being $ac = \mathsf{unlinkArtifacts}(ar_{id}, ar'_{id})$ the action to do in order to remove the link between ar_{id} and ar'_{id}, and being $ev = \langle \mathsf{unlink_req}, \langle ag_{id}, req_v, t\rangle\rangle$ then the action succeeds and:

$$\frac{ev \in Ev \quad \mathsf{triggering}(ev, R) = \bot \quad \langle ar_{id}, ar'_{id}\rangle \in Lm}{\langle Ag, Ar, Art, Ev \cup ev, M, R, t\rangle \longrightarrow \langle Ag, Ar, Art, Ev', M', R, t'\rangle}$$

where:

- $Ev' = Ev \cup \langle \mathsf{ar_unlinked}, \langle ar_{id}, ar'_{id}, t\rangle\rangle$
- M' contains an updated linkability map, $Lm' = Lm \setminus \{\langle ar_{id}, ar'_{id}\rangle\}$

On the agent side $ag'_{Ev} = \{\langle \mathsf{action_completed}, \langle \mathsf{unlinkArtifacts}(ar_{id}, ar'_{id}), \bot\rangle\rangle\}$.

Exploring Environment

An action is provided to agents in order to retrieve in the workspace existing artifacts of a specified type:

$$\mathsf{lookupArtifact}(art_n)$$

The action accounts for retrieving the set of identifiers of all artifacts of type art_n actually existing in the workspace. Formally:

Definition 21 *(artifact lookup)* Let $\langle Ag, Ar, Art, Ev, M, R, t\rangle$ be the configuration of the workspace and $\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle \in Ag$ the configuration of the agent. Being $ac = \mathsf{lookupArtifact}(art_n)$ the action selected by the agent, and being $ev = \langle \mathsf{lookup_req}, \langle ag_{id}, req_v, t\rangle\rangle$, then the action completes in the same transition:

$$\frac{ev \in Ev \quad \mathsf{triggering}(ev, R) = \bot \quad \langle art_n, art_{init}, art_{pr}, man\rangle \in Art}{\langle Ag, Ar, Art, Ev \cup ev, M, R, t\rangle \longrightarrow \langle Ag, Ar, Art, Ev', M, R, t'\rangle}$$

where $Ev' = Ev \cup \langle \mathsf{ar_looked}, \langle ag_{id}, art_n, t\rangle\rangle$ and the added event indicates that a look up action has been performed by agent ag_{id} for art_n – at time t.

On the agent side, the set of events is updated according to:

$$\frac{ev \in Ev \quad \mathsf{triggering}(ev, R) = \bot \quad \langle art_n, art_{init}, art_{pr}, man\rangle \in Art \quad Ar_{id} \subset Ar}{\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle \longrightarrow \langle ag_{id}, ag_s, ag'_{Ev}, ag_{pr}\rangle}$$

where $ag'_{Ev} = \{\langle \mathsf{action_completed}, \langle \mathsf{lookupArtifact}(art_n), Ar_{id}\rangle\rangle\}$ and the action feedback is the set Ar_{id} of all the artifact identifiers ar_{id} such that $\langle ar_{id}, ar_t, I, P, V, O\rangle \in Ar$.

Remark: The lookupArtifact in CArtAgO implementation retrieves the (system) identifier of an artifact given its logic name. This is not included in the abstract model since a unique identifier ar_{id} is used for both the logic name and the system identifier.

Inspecting Artifacts

An action is provided to consult the manual of a specific type of artifacts:

$$\mathsf{consultMan}(ar_t)$$

that is assumed to improve the agent knowledge about the artifact type ar_t, and in turn improve the agent repertoire with the actions needed to exploit instances of the specified artifact type. Formally:

Definition 22 *(artifact manual consult)* Let $\langle Ag, Ar, Art, Ev, M, R, t\rangle$ be the configuration of the workspace and $\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle \in Ag$ the configuration of the agent. Being $ac = \mathsf{consultMan}(ar_t)$ the action to do such that $\langle art_n, art_{init}, art_{pr}, man\rangle \in Art$, and being $ev = \langle \mathsf{consult_req}, \langle ag_{id}, req_v, t\rangle\rangle$ the event indicating the request to consult the manual for an existing artifact type done by agent ag_{id} at time t – req_v identifies the consultMan action, including the artifact type ar_t for the artifact to be consulted, then the action completes in the same transition:

$$\frac{ev \in Ev \quad \mathsf{triggering}(ev, R) = \bot \quad \langle art_n, art_{init}, art_{pr}, man\rangle \in Art}{\langle Ag, Ar, Art, Ev \cup ev, M, R, t\rangle \longrightarrow \langle Ag, Ar, Art, Ev', M, R, t'\rangle}$$

where the added event indicates that a manual consult action ha been performed by agent ag_{id} for ar_t – at time t: namely, $Ev' = Ev \cup \langle \mathsf{ar_consulted}, \langle ag_{id}, ar_t, t\rangle\rangle$.

On the agent side:

$$\frac{ev \in Ev \quad \mathsf{triggering}(ev, R) = \bot \quad \langle art_n, art_{init}, art_{pr}, man\rangle \in Art}{\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle \longrightarrow \langle ag_{id}, ag_s, ag'_{Ev}, ag'_{pr}\rangle}$$

where the action feedback is the value of the manual man, and the ag_{pr} is improved with the new knowledge represented in man. In particular:

- $ag'_{Ev} = \{\langle \mathsf{action_completed}, \langle \mathsf{consultMan}(ar_t), man\rangle\rangle\}$. In this case the action feedback is the man value.
- $ag'_{pr} = ag_{pr} \cup \mathsf{translate}(man)$,.

Remark: In CArtAgO implementations, the translate function is realized within the integration technology allowing heterogenous agents to play in the work environment. Here the function translate\1 is assumed to translate the informational content brought by the manual in knowledge processable in terms of the specific agent model that has been integrated.

5.3.6 Workspace Time Evolution

Finally, a transition is used to represent the discrete evolution of the logic time inside the workspace:

$$\langle Ag, Ar, Art, Ev, M, R, t\rangle \longrightarrow \langle Ag, Ar, Art, Ev, M, R, t+1\rangle$$

Remark: In concrete implementation, the transition is triggered by the internal clock of the processor which is executing the workspace.

5.4 Final Remarks on the Formalisation

This chapter presented a formalisation of a general-purpose model for programming environments in MAS based on the notion of agents, artifact and workspaces as proposed by the A&A approach implemented by the CArtAgO framework.

The abstract model is useful to understand and analyze the features that artifact-based environments have for environment programming and – more generally – for programming MAS. A first remarkable feature of the model is that it allows for the concurrent evolution of agent and artifact configurations which are not related by some dependency. This promotes the parallel execution of operations affecting distinct artifacts inside the workspace.

A second property is related to the synchronization between an agent and an artifact. it happens when an agent executes a use action, which completes only when (if) the triggered operation completes. However this does not necessarily block the evolution of the agent configuration: this happens only if the program specifies that every further agent action – including reactions to environment events – must be done after the completion of the executed operation. No synchronization occurs instead when an artifact triggers the execution of a link operation over another artifact.

An important result of the proposed model refers to time management: exploiting the fact that a single workspace is not distributed, we could define a single (logical) notion of time at the workspace level. As showed in this chapter, for each workspace it is feasible to assume that *(i)* a local logical notion of time can be defined, and *(ii)* observable events occurring the in the workspace can be totally ordered using logical timestamps, even if they are generated by different artifacts running concurrent processes. Given this assumption, agents perceive chains of events, which are totally ordered if the source is a single workspace, partially ordered if more sub-environments are involved.

Distribution is handled by considering multiple workspaces, each with its own logical time and possibly running on a different network node. Although not captured by the model – which is about a single workspace – artifacts belonging to different workspaces can be linked together, so that an operation executed on an artifact located in a workspace can have an effect on an artifact located in a distinct workspace, eventually running on a different network node.

The set of artifact types available in a workspace is defined by the initial configuration, as well as the set of agents working inside the workspace. Actually, the model could be easily extended by improving the repertoire of agents with new actions aimed to manage the set of artifact types available in a workspace, thus enabling agents to add, remove or update a type dynamically.

It is worth to remark that the model has been conceived so to be easily integrated with existing agent formalizations such as in [31, 14, 139]. This enables future investigation including aspects actually missed in the formal description, as for instance the ones related to dialogic interactions among agents (as provided in [36, 130]). The adopted approach also accounts for the formal verification of global properties of the system built in terms of agents and artifacts, i.e. based on existing research work as [37].

The presented model revises and extends an already presented formalisation [124] and includes a new event based execution model which has been already implemented in the last version of CArtAgO. The meanings of the envisaged changes will be clarified in the next chapter, where a new construct will be introduced to regulate intra-workspace dynamics. On the basis of the event based mechanism detailed here, the new construct accounts the possibility to replace the default transitions with programmable ones.

Chapter 6

Extending CArtAgO with Intra-Workspace Dynamics

As in Chapter 5 a formal description for an A&A based system has been provided, focusing in particular on the dynamics at the basis of interactions between agents and artifacts, in this chapter a complementary mechanism for specifying global dynamics inside the workspace is introduced. The mechanism involves the definition of event driven rules governing primer mechanisms inside a workspace and is aimed at providing a further, pervasive management for the work environment. Thereby, environment dynamics can be programmed not only on the basis of the behavior of the single agents interacting with artifacts and workspaces, but also at a macro level, i.e., defining global workspace laws. This is done by introducing general rules, triggered by environment events, capable to manipulate the space of events and affect workspace configurations, i.e. triggering operation on artifacts, creating and disposing artifacts, enabling and disabling operations, etc.

6.1 Specifying global dynamics inside workspaces

Table 6.1 resumes the configuration of a Multi-Agent System based on the first class abstraction defined in the in the previous chapters, namely agents, artifacts and workspaces. In this configuration a MAS can be composed by a series of workspaces, each containing a set of heterogeneous agents (denoted by Ag, which structure has been placed in Subsection 5.2.1) and a set of artifacts (denoted by Ar, as described in Subsection 5.2.2). Along with the unambiguous specification

$$
\begin{aligned}
MAS &= \langle Ws \rangle \\
Ws &= \{\langle ws_n, \langle Ag, Ar, Art, Ev, M, R, t \rangle \rangle\} \\
Ag &= \{\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr} \rangle\} \\
Ar &= \{\langle ar_{id}, ar_t, I, O, P, V \rangle\}
\end{aligned}
$$

Table 6.1: Structures of a Multi-Agent Systems based on A&A, as it follows from Subsection 5.2.5.

of basic interactions taking place inside the workspace between agents and artifacts, and between agents and workspaces, a new mechanism is introduced in this chapter in order to control global dynamics inside the environment. The proposed approach involves the definition of event driven rules, which execution is aimed at governing primer mechanisms inside a workspace and at providing the environment developer with a further, pervasive mechanism of management and control. Such rules, referred as workspace rules, are specified applying sequences of operators to the entities actually dwelling the workspace. Their execution is then aimed at eliciting additional outcomes once specific events occurs, thus altering the normal dynamics inside the environment.

After having introduced the general principles underlying intra-workspace dynamics in the next sections, Section 6.3 and Section 6.4 formally describe respectively a programming language allowing to specify workspace rules and the effects of applying them to the environment dynamics. The proposed approach is enriched by the description of concrete programming examples Section 6.5 and general remarks rooted in the context of related work Section 6.6.

6.2 Shaping the problem

The definition of global environment dynamics is deemed as an important aspect of environment programming in the context of this work. It is motivated by the need to offer the developer with a different level of programming computational entities used as facilities instrumenting the work environment. Imagine for instance an A&A context requiring to face with the following scenery:

- The developer has the need to specify the access control to artifacts at a fine granularity, namely by specifying different rights to use artifacts for each operation defined in their usage interfaces.
- The developer has the need to redirect particular interactions, namely to

replace operation requests once an artifact is overloaded, or to create new instances of a given artifacts once the one actually in place is is blocked.

- The developer has the need to enable or disable the use of specified artifact operations for a specified agent, once its behavior is supposed to be suspicious.

In cases like the abovementioned, an environment framework should provide mechanisms to adapt global dynamics of the environments based on the actual needs, thus allowing the developer to use some grade of freedom in specifying, at design time, alternative functioning inside the workspace once particular situations occurs. In such a case, in order to avoid undesired outcomes and maintain the global coherence of the system, the developer may want the use of certain artifacts to be prevented, or he/she may want to redirect the related operation requests to an alternative resources, and so on.

To this end, we argue that the framework should provide some effective mechanism to control environment dynamics at a macro level, namely besides the specification of the interactions which may occur at the micro level between the single decentralised structures (artifacts) and the single individuals (agents). This kind of management requires the capability for a direct and powerful intervention on the overall set of environment entities. This capability should allow to control the involved entities at a global level, while its deployment and the management should be as much automated as possible, i.e., without requiring a human intervention, nor to stop the system and change its programs.

6.2.1 Programming approaches

In order to meet these requirements in the context of an A&A based model, different approaches can be adopted. A first approach may involve the introduction of specific agents to which the task to govern environments could be delegated. This option envisages the adoption of environmental agents, namely worker agents which task is to regulate and adapt internal workspaces dynamics according to defined policies. Those agents would have in charge the task to regulate the work environment, thus having access to an additional set of operations aimed at modifying the workspace structures—working inside the systems with the capabilities deemed for a sort of super user. Although the adoption of an agent with these characteristics is still possible according to the programming model provided in the previous chapters, this choice conflicts with the dictates of A&A meta-model, according to which an agent abstraction does not properly address the task to provide

environment functioning. In fact, managing environment dynamics does not need traditional properties of agency like autonomy, goal orientation, encapsulation of control, social ability, proactiveness, etc. In A&A terms, agents are assumed to *exploit* environment facilities more than autonomously govern their global functioning. More than being associable to agent "black boxes" hiding their internal specification and providing an autonomous behavior, environment dynamics should be governed by fixed laws. The specification of such laws should be inspected, possibly learned and also changed by other participating agents. Namely, specifying environment dynamics should be an agile task for system administrators (either computational either human ones): global laws should be changed on the fly, while introducing changes in environment dynamics should be allowed either at design time and at run time.

A second approach may involve the adoption of embedded artifact functionalities to govern the whole environment. This option envisages the sole use of artifacts in order to automatically manage global environment dynamics. This is possible according to the computational model of artifacts, adopting, for instance a massive use of link operations, that, as said, enables functional relations between different artifacts. In this case, the specific events or states occurring inside artifacts may be propagated across links and then handled from artifact to artifact, as described in previous chapters. On the other hand, artifacts in A&A are viewed as decentralised entities, namely building blocks instrumenting work environments with special purpose functionalities. Making a massive use of linking between artifacts has the drawback to "pollute" artifact computational model: namely, involved artifacts should be programmed not only to serve the purpose for which they are conceived at design time, but also to manage operation links, that is to relate and possibly react to several events occurring in other part of the environment. This, in practice, drastically augment the coupling between different artifacts as the resources needed both to request and serve link operations have to be embedded inside an artifact program. Accordingly this option implies the need to provide additional logics to artifacts, making them tight coupled to the application domain, thus resulting in highly compromising some wide adopted principles of software engineering as reuse, separation of concerns, data hiding etc.

An alternative approach has been proposed in the context of coordination models as Agent Coordination Context (ACC), a runtime tool to control agent-environment interaction [92]. ACC has been inspired by the notion of "cockpit", or control panel, which has been be reified inside a MAS as a programmable interface exploitable by agents to interact with environments. It works at further mediating interactions, hence regulating, from time to time, what actions the agent can

do and what perceptions are received. This allows an high separation of coordination concerns: from an agent/subjective perspective an ACC can rule individual agent (inter)actions; from a system/objective perspective, an ACC can be used to enforce global laws of the systems, i.e. providing access control, redirecting actions to a different target resource etc.

Whereas the approach is suitable to control micro level interactions involving agents and environment resources, specifying global laws would be not an easy task. A possible drawback is here the need to manage many decentralised ACCs at runtime, namely one for each agent entering the system. This requires to program a pool of ACCs, each responding at the same rules and concurrently operating on the same environment resources. In practice, this approach shift the problem of coordinating a set of environment entities to coordinating a set of ACCs.

In this work, in order to leave the management of global environment dynamics outside agents and artifacts abstractions, a different approach is envisaged. The idea is to provide the macro level dynamics inside the environment as an orthogonal aspect with respect to the management of micro level interactions. As a consequence, the approach is deemed to let global management of environment outside the scope of agents and artifacts. Otherwise, we argue that specifying such mechanisms has to be explicitly defined as a separate task, rather than entangling it with the programming model defining artifacts and agents. The approach envisages the introduction of global laws which can be viewed as a sort of "connective tissue" for the overall work environment. On these basis, the workspace machinery is enriched with a separate mechanism, governed by a rule based engine. Such rules define "physic" laws of the system and are introduced as a programmable part of a workspace aimed at ruling over the overall interaction space.

6.2.2 Workspace Rules

An important assumption made in the previous chapter was the pivotal role played by events in defining the workspace dynamics: as showed by the operational semantics described in Chapter 5, in a workspace an event is always a signal of change (i.e. the request made by some agent to use an operation, the execution of an operation step, the change of some observable property, an agent joinin/leaving the workspace, etc.). As far as the computational model has been conceived, workspace events are used to trigger workspace transitions, while chains of events can be sequenced to trace series of state transitions.

Workspace Rules are programming constructs allowing to specify alternative relations between events, hence they are aimed at manipulating the workspace

in the domain of events. This can be done by replacing the default transitions (as they have been envisaged in the previous chapter) with programmable transitions. Taking the physical metaphor, since the occurrence of a certain action typically performed by an agent, a workspace rule provides a mean to trigger a programmable reaction, which in turns is aimed at applying a set of "forces" capable to affect other structures inside the environment[1].

In the characterization of rules described by John Searle [127, 128] workspace rules – applied inside a MAS work environment – resemble the notion of *regulative rules*, namely rules aimed at regulating antecedently existing forms of behavior. As in the case of regulative rules, workspace rules settle pre-existing functionalities—when the existence of these functionalities is logically independent from the existence of the rules themselves. As in the case of regulative rules, the regulatory mechanism of a workspace rule is given in an imperative fashion. As will be described in Section 6.3, workspace rules are specifiable as event-condition-action (ECA) rules, having the form "if *ev* in the context *c* apply *a*" or "when *ev* in the context *c* apply *a*".

Figure 6.1 shows an example of workspace regulated by workspace rules. The figure represents a workspace populated by agents using and observing artifacts. Some of the events generated on the basis of such interactions are in this case intercepted by workspace rules, which in turns trigger the execution of reactions further addressed at acting upon environment structures

Different kind of operators can be specified along with workspace rules. Table 6.2 shows a meaningful set of possible workspace rule operators. We here consider, for simplicity, those operators which are worth to be taken into account hereinafter in the context of this work, while the definition of additional operators is demanded to future works. A first type of operators in Table 6.2 involves constructs allowing to trigger the execution of operations on specified artifact inside the workspace (1-2 in Table 6.2). The second group includes operators to create or remove artifacts in the actual workspace configuration (operators 3-4). The third group considers operators aimed at preventing or enabling the use of particular operations in a specified artifact (operators 5-6). Finally, two operators are specified to take out or include an agent from the actual workspace configuration (operators 7-8) .

Some assumptions are needed to define the scope of applicability of the work-

[1]The adoption of the term "forces" stresses the analogy with classical physics: workspace rules may resemble, in the computational context of an A&A system, what Newton's laws of motion represent in explaining mechanics [86], or Kepler's laws of planetary motion are in explaining laws of universal gravitation [74].

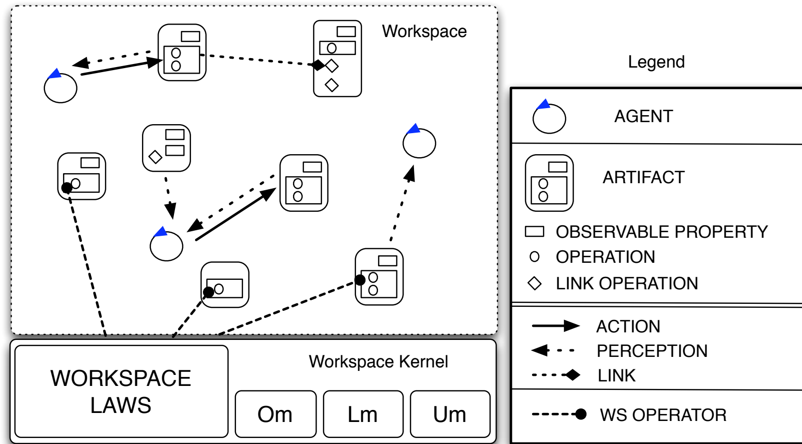

Figure 6.1: Example of workspace dynamics regulated by workspace rules.

(1)	`applyOp`(ar_{id}, op_{name} $[, Params]$)
(2)	`applyLop`(ar_{id}, op_{name} $[, Params]$)
(3)	`make`(ar_{id}, art_n $[, Params]$)
(4)	`dispose`(ar_{id})
(5)	`disable`(ar_{id} $[, ag_{id}]$ $\{, op_{name}\}$)
(6)	`enable`(ar_{id} $[, ag_{id}]$ $\{, op_{name}\}$)
(7)	`exclude`(ag_{id})
(8)	`include`(ag_{id})

Table 6.2: Basic operators specifiable as workspace rule reactions in CArtAgO.

space rules. In particular we need to make assumptions about time management, rules execution and possible failures in order to prevent inconsistencies and guarantee the global coherence of the system.

Time By exploiting the fact that a single workspace is not distributed across different nodes, it was possible to define a single notion of (logical) time at the workspace level. As explained in Chapter 5, this allowed events inside a workspace to be marked according to the specification of a timestamp. The time model adopted allows for events to occur at the same logical time.

Atomic Execution The reactions specified in the body of a workspace rule are executed sequentially as they are specified in the body. The execution of the

whole set of reactions in the body is envisaged to be atomic, hence all the reactions specified within the body of the rule, as well as any of the multi step operations triggered upon artifacts, have to complete before allowing further activities inside the workspace. Accordingly, all the involved resources inside the workspace are locked until the execution of the rule completes: in order to prevent concurrency and inconsistencies with other ongoing activities, all the involved artifacts are locked until the completion of the all the reactions specified in the body.

Failures The execution of the body of a workspace rule can complete with success or failure according to the execution of the involved reactions. In particular, the triggered rule has to be considered failed as soon as the application of one of the operators specified in the body of the rule fails. In order to prevent inconsistencies, in case of failure the global state of the workspace has to be reconfigured as it was before the partial execution of the rule, at the time in which the rule was triggered.

Agent Autonomy Workspace Rules are conceived to specify and enact particular dynamics inside the workspace. Those dynamics are applied to the entities actually dwelling the workspace without affecting agent's autonomy.

In what follows a language for specifying workspace rules is provided firstly introducing a formal syntax and then providing a description of the execution model in terms of operational semantics.

6.3 Sintax

Workspace Rules are programs collected at the level of the single workspace and stored in what we placed as the set R in the workspace configuration (see Subsection 5.2.3 for the complete description of a workspace configuration). The defined rules can intercept specified events collected at the workspace level (actually, events added in the workspace Ev set). The triggering events are specified in the left-hand side of the workspace rule. Whether a specified context condition holds, the right-hand side of the rule rule can be applied and the execution of the related script is triggered. In so doing, workspace rules can be specified as *reactions* to workspace events, each reaction specifying a list of operator constructs aimed at modifying the workspace event set and, as a consequence, at practically elicit changes in the global configuration of the workspace configuration.

```
⟨R⟩             = {⟨w_entry⟩}
⟨w_entry⟩       = ⟨r_id, w_rule⟩
⟨w_rule⟩        = ⟨trigg_ev⟩[":"⟨context⟩] "->" ⟨body⟩
-------------------------------------------------------------------------
⟨trigg_ev⟩      = "+"⟨ev⟩
⟨ev⟩            = ⟨ev_t⟩ "(" ⟨ev_v⟩ ")"
⟨context⟩       = "true" | ⟨ctx_exp⟩ | "(" ⟨ctx_exp⟩ ")"
                | ⟨ctx_exp⟩ "&" ⟨ctx_exp⟩ | ⟨ctx_exp⟩ "|" ⟨ctx_exp⟩
⟨body⟩          = "true" | ⟨wr_operator⟩ [ ";" ⟨wr_operator⟩] "."
-------------------------------------------------------------------------
⟨ev_t⟩          = "op_req" | "linkop_req" | "focus_req" | "obs_req"
                | "stopFocus_req" | "link_req" | "unlink_req"
                | "make_req" | "dispose_req" | "lookup_req"
                | "join_req" | "leave_req" | "consult_req"
                | "op_signal" | "op_ongoing" | "op_completed"
                | "op_failed" | "ar_created" | "ar_disposed"
                | "prop_observed" | "prop_updated" | "prop_new"
                | "prop_removed" | "ar_looked"| "ar_consulted"
                | "ar_focused" | "ar_unfocused" | "ar_linked"
                | "ar_unlinked" | "ws_joined" | "ws_leaved"
⟨ev_v⟩          = ⟨list_of_terms⟩
-------------------------------------------------------------------------
⟨ctx_exp⟩       = ⟨wr_exp⟩ | "not" ⟨wr_exp⟩
                | ⟨wr_exp⟩ ⟨ar_op⟩ ⟨wr_exp⟩
⟨wr_exp⟩        = ⟨prop_exp⟩ | ⟨term⟩
⟨prop_exp⟩      = ⟨term⟩ ":" ⟨obs_prop⟩
⟨obs_prop⟩      = ⟨p_n⟩ "(" ⟨p_v⟩ ")"
⟨p_n⟩           = ⟨term⟩
⟨p_v⟩           = ⟨list_of_terms⟩
-------------------------------------------------------------------------
⟨wr_operator⟩   = "skip" | ⟨applyOp⟩ | ⟨applyLop⟩ | ⟨make⟩
                | ⟨disable⟩ | ⟨enable⟩ | ⟨exclude⟩ | ⟨include⟩
⟨applyOp⟩       = "applyOp(" ⟨list_of_terms⟩ ")"
⟨applyLop⟩      = "applyLop(" ⟨list_of_terms⟩ ")"
⟨make⟩          = "make(" ⟨list_of_terms⟩ ")"
⟨dispose⟩       = "dispose(" ⟨list_of_terms⟩ ")"
⟨disable⟩       = "disable(" ⟨list_of_terms⟩ ")"
⟨enable⟩        = "enable(" ⟨list_of_terms⟩ ")"
⟨exclude⟩       = "exclude(" ⟨list_of_terms⟩ ")"
⟨include⟩       = "include(" ⟨list_of_terms⟩ ")"
-------------------------------------------------------------------------
⟨ar_op⟩         = ">" | ">=" | "<" | "<=" | "==" | "!=" | "mod" | "div"
⟨list_of_terms⟩ = ⟨term⟩ {","⟨term⟩}
⟨term⟩          = ⟨atomic_formula⟩ | VAR | NUMBER | STRING
⟨atomic_formula⟩ = ATOM [ "("⟨list_of_terms⟩")" ]
```

Table 6.3: Syntax of workspace rules in CArtAgO

Table 6.3 shows the syntax description for the language used to specify workspace rules. Non terminal symbols are enclosed by $\langle\rangle$. A Workspace Rules is an entry in the set R specified in a workspace configuration. Each rule is composed by a triggering event $\langle trigg_ev\rangle$ possibly followed by a context condition $\langle context\rangle$ (forming together the activation condition, i.e. the antecedent of the workspace rule) and by a $\langle body\rangle$ specifying the sequence of operators to be applied (representing the consequent of the workspace rule). Each triggering event is preceded by a "+" symbol, and is composed by an event type and by an event value. Elements $\langle ev_t\rangle$ belong to a finite set of event types and in Table 6.3 are listed according to the events processed inside a workspace (as listed in the tables of Section 5.2). Event values $\langle ev_v\rangle$ are list of terms including the informational content related to the event. Context expressions are first order formulae including expressions related to artifact observable properties ($\langle prop_exp\rangle$). Each property expression can be specified since an artifact identifier, followed by the terminal symbol ":" and by an observable property $\langle obs_prop\rangle$. Observable properties are then defined since a property name and value ($\langle p_n\rangle$ and $\langle p_v\rangle$), which refer respectively to terms and list of terms. The list of applicable operators $\langle wr_operator\rangle$ can include one of the operators envisaged in Table 6.2 ("skip" means no operator applied). Finally, $\langle list_of_terms\rangle$ can be sequences of atomic formulae, variables, numbers as well as integers or floating point, and strings. Non-terminals symbols as $ATOM, VAR, NUMBER$ and $STRING$ are not included in the specification for brevity: they refer to the corresponding data types as used in Prolog, namely *predicates, variables, numbers* and literal *compound terms*.

After having discussed the semantic aspects related to workspace rules, Section 6.5 shows programming examples of their application.

6.4 Dynamics

Before describing the transition involving workspace rules, some definition need to be specified. In particular, the definition for the global workspace observable state Ar_P and for the triggering function are provided. In what follows the operator $\models$ is used to model a first order entailment relation according to most general unifier function — the concrete implementation of this relation makes use of a Prolog like engine to provide unification of variables.

We define the global workspace observable sate in terms of artifacts observable properties, namely indicating environment states through what agents may observe in the overall workspace. Formally:

Definition 1 *(workspace observable state)* Let $\langle Ag, Ar, Art, Ev, M, R, t \rangle$ be the configuration of the workspace. For each artifact $ar \in Ar$ we define the overall workspace observable sate as:

$$Ar_P = \bigcup_{ar \in Ar} P_{ar}$$

In other words, the workspace observable state is the summation ot the entire set of the observable properties contained in that workspace, considered for each single artifact in Ar.

For each event collected in the workspace in Ev, we also define a triggering function in terms of the effects that the event has on the set R of workspace rules. In particular, given the configuration of R and given an event in Ev, the triggering function returns the entry of the rule that matched the event, if any, and $\perp$ otherwise. Formally:

Definition 2 *(triggering function)* Let $\langle Ag, Ar, Art, Ev, M, R, t \rangle$ be the configuration of the workspace, Ar_P the set of all observable properties for the artifacts in Ar. Let `trigg_ev :  context -> body` be the specification of a *w_rule* being $\langle r_{id}, w_rule \rangle$ an associated entry in R. Being *ev* be an event in Ev, being *trigg_ev* specified in R and being $Unify(ev, trigg_ev)$ a function that returns the most general unifier of *ev* and *trigg_ev* if they are unifiable, otherwise it returns $\perp$, then the function triggering(ev, R) is defined as follows:

$$\mathsf{triggering}(ev, R) = \begin{cases} \langle r_{id}, w_rule \rangle, & \text{if } \exists \quad \tau_1, \tau_2 : \quad Unify(ev, trigg_ev) = \tau_1 \\ & \qquad \& \quad Ar_P \models context\, \tau_1 \tau_2 \\ & \& \ \nexists ev' : ev' = ev, \quad \mathsf{triggering}(ev', R) \neq \perp \\ & \\ \perp, & \text{otherwise} \end{cases}$$

Namely, if exist substitutions τ_1 and τ_2, so that $Unify(ev, trigg_ev) = \tau_1$ and $Ar_P \models context\, \tau_1 \tau_2$, and if the same event has not been applied formerly for triggering a workspace rule, then $\mathsf{triggering}(ev, R) = \langle r_{id}, w_rule \rangle$. Otherwise the function is not defined and $\mathsf{triggering}(ev, R) = \perp$.

Since the event *ev* triggering a workspace rule has to be reapplied to the system to trigger further transition not involving workspace rules, we need to check whether the event has not been applied previously to trigger the same rule. This control is carried out by the triggering function: as far as this function has been defined, the condition $\mathsf{triggering}(ev', R) \neq \perp$ prevents the application of the same event for multiple execution of the same rule. The effect of this condition, in

particular, is twofold: first, it prevents multiple application of the same rule for the same event; second, it prevents the application of multiple rules for the same event.

Notice also that the substitution τ_1 is applied to the *context* condition of the rule before testing it against the workspace observable state Ar_P. This results in a second substitution τ_2 that, together with τ_1, is then applied to the body of the rule before executing it. The application of the substitutions to the *context* and the *body* of the rules ensures that the unified value of the possible shared variables are passed from the head of the rule ($trigg_ev$) to the context condition (*context*), and finally to the body reactions.

When an event *ev* in the workspaces triggers a workspace rule, the context condition is evaluated against the workspace observable state and, if the condition holds, the body of the rule is executed atomically. All the specified operators are executed in a non interleaved way, and, in case of success, the overall systems changes in a single transition after having applied the entire set of operators in the order specified in the body.

In the following sections, the effects of the transitions elicited by workspace rules are formally described. The provided operational semantic focuses on the different entities of the system involved, respectively taking into account the transitions for the configuration of workspace, artifacts and agents.

Workspace Rules on Workspace

Definition 3 *(workspace rules)* Let $\langle Ag, Ar, Art, Ev, M, R, t\rangle$ be the configuration of the workspace. Let `trigg_ev : context -> body` be the specification of a $\langle w_rule\rangle$ where all the variables are fresh, and let be $\langle r_{id}, \langle w_rule\rangle\rangle$ an associated entry in R. Being ev an event in Ev, and being triggering(ev, R) a function returning $\langle r_{id}, \langle w_rule\rangle\rangle$ if exist in R an entry unifying the event ev, and $\perp$ otherwise. Then, the successful execution of w_rule will elicit the following transition in the workspace:

$$\frac{ev \in Ev \qquad \text{triggering}(ev, R) = \langle r_{id}, \langle w_rule\rangle\rangle}{\langle Ag, Ar, Art, Ev \cup ev, M, R, t\rangle \longrightarrow \langle Ag', Ar', Art, Ev' \cup ev, M', R, t'\rangle}$$

The new set of agents Ag' is updated for each exclude reaction included in the body of the triggered rule, in particular:

- $Ag' = Ag \setminus \{\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle\}$ for each exclude(ag_{id}) specified in *body*;

- $Ag' = Ag \cup \{\langle ag_{id}, ag_s, ag_{Ev}, ag_{pr}\rangle\}$ for each $\mathsf{include}(ag_{id})$ specified in *body*.

Besides, the new set of artifacts Ar' is updated for each make and for each dispose reactions included in the body of the triggered rule, in particular:

- $Ar' = Ar \cup \{\langle ar_{id}, ar_t, I, P, V, O\rangle\}$ for each $\mathsf{make}(ar_{id}, ar_t, Params)$ included in *body*;

- $Ar' = Ar \setminus \{\langle ar_{id}, ar_t, I, P, V, O\rangle\}$ for each $\mathsf{dispose}(ar_{id})$ included in *body*.

The new set of workspace maps M' is updated for each enable and for each disable operators included in the body of the triggered rule. Those operators act by updating the usability map, in particular by adding and removing entries in M_{Um} thus indicating whether some agents has (or has not) the rights to use a given artifact operation op. By default, every agent has the rights to use any given artifact operation. An entry In particular:

- $M'_{Um} = M_{Um} \cup \{\langle ar_{id}, ag_{id}, op_{name}\rangle\}$ for each operation specified by the operator $\mathsf{disable}(ar_{id}, ag_{id}, \{op\})$ included in *body*;

- $M'_{Um} = M_{Um} \setminus \{\langle ar_{id}, ag_{id}, op_{name}\rangle\}$ for each operation specified by the operator $\mathsf{enable}(ar_{id}, ag_{id}, \{op\})$ included in *body*.

Finally, the new set of events $Ev' \cup ev$ includes both the updated set Ev' and the triggering event ev. ev has to be kept in order to be processed by possible other transitions inside the workspace, given that the function $\mathsf{triggering}(ev, R)$ won't trigger the same workspace rule for two times. Ev' is updated including all the events related to the execution of the reactions included in the body of the triggered rule. In particular, for each operation triggered by a reaction in the body the following events are added:

- $Ev' = Ev \cup \langle \mathsf{op_signal}, \langle ar_{id}, s_t, s_v, t\rangle\rangle$, indicating that a signal $\langle s_t, s_v\rangle$ is launched by artifact ar_{id} at time t during the execution of the operation step;

- $Ev' = Ev \cup \langle \mathsf{op_completed}, \langle ar_{id}, op_{req}, t\rangle\rangle$, indicating that a triggered operation succeeded at time t.

Besides, for each change in observable properties elicited by some reaction in the body of the triggered rule, Ev' changes as it follows:

- $Ev' = Ev \cup \langle \mathsf{prop_updated}, \langle ar_{id}, p_n, p_v, t\rangle\rangle$ indicating that a property p_n has been updated in artifact ar_{id}, at time t;

- $Ev' = Ev \cup \langle \text{prop_new}, \langle ar_{id}, p_n, p_v, t \rangle \rangle$ indicating that a property p_n has been added in artifact ar_{id}, at time t;
- $Ev' = Ev \cup \langle \text{prop_removed}, \langle ar_{id}, p_n, p_v, t \rangle \rangle$ indicating that a property p_n has been removed in artifact ar_{id}, at time t.

In addition, for each change in the set Ar of artifacts in the workspace elicited by some reaction in the body of the triggered rule, a related event is added to Ev:

- $Ev' = Ev \cup \langle \text{ar_created}, \langle \bot, ar_{id}, at_t, t \rangle \rangle$ indicating that an artifact ar_{id} has been created, at time t;
- $Ev' = Ev \setminus \langle \text{ar_disposed}, \langle \bot, ar_{id}, at_t, t \rangle \rangle$ indicating that an artifact ar_{id} has been disposed, at time t.

Finally, for each change in the set Ag of agents in the workspace elicited by some reaction in the body of the triggered rule, a related event is added to Ev, in particular $Ev' = Ev \cup \langle \text{ws_leaved}, \langle ag_{id}, t \rangle \rangle$, indicating that the agent ag_{id} leaved the workspace at time t.
Remarks: (a) As far as workspace rules have been defined, the reactions specified in a body of a workspace rule have priority above all the other transitions. In addition, the body of a triggered workspace rule is executed atomically and all the involved artifacts are locked until the completion of the body. As a consequence, all the operations triggered by single reactions must complete before the completion of the execution of the body and no pending operation steps are allowed before unlocking the control of artifacts. Notice, in this case, the absence of op_ongoing events at the end of the transition elicited by a workspace rule. **(b)** In reactions involving creation or disposal of existing artifacts in Ag, the first element of the event value (ev_v) is set to $\bot$, indicating in this case that the artifacts are created or disposed by the internal dynamics of the workspace, in spite of the case where they are created or disposed by some agent acting in the workspace.

Handling failures

Once a failure is generated during the execution of a rule body, the system has to ensure that the configurations are rolled back to the previous consistent state.

Definition 4 *(workspace rules - failures)* Let $\langle Ag, Ar, Art, Ev, M, R, t \rangle$ be the configuration of the workspace. Let `trigg_ev : context -> body` be the specification of a $\langle w_rule \rangle$ where all the variables are fresh, and let be $\langle r_{id}, \langle w_rule \rangle \rangle$

an associated entry in R. Being ev an event in Ev, and being $\mathsf{triggering}(ev,R)$ the function returning $\langle r_{id}, \langle w_rule \rangle \rangle$ if exist in R an entry unifying the event ev, and $\perp$ otherwise, then, the unsuccessful execution of w_rule will is described by the following transition:

$$\frac{ev \in Ev \qquad \mathsf{triggering}(ev,R) = \langle r_{id}, \langle w_rule \rangle \rangle}{\langle Ag, Ar, Art, Ev \cup ev, M, R, t \rangle \longrightarrow \langle Ag, Ar, Art, Ev, M, R, t' \rangle}$$

that is, the workspace structures are maintained unchanged as they were before the application of the workspace rule.

Remarks: (a) In CArtAgO the failure management is realized by making a copy of all the entities involved in the execution of a workspace rule. In order to restore the system, as soon as a failure is registered, the system makes a roll back to the previously consistent state: In other terms, once a failure occurs during the execution of a workspace rule the copy is restored in order to re-establish a consistent configuration.

Workspace Rules on Artifacts

Artifacts are affected by the execution of workspace rules if they are involved in one or more reactions specified in the body of the rule.

Definition 5 *(workspace rules - artifact)* Let $\langle Ag, Ar, Art, Ev, M, R, t \rangle$ be the configuration of the workspace and $\langle ar_{id}, ar_t, I, P, V, O \rangle \in Ar$ the configuration for an artifact $ar_{id} \in Ar$. Let `trigg_ev : context -> body` be the specification of a $\langle w_rule \rangle$ where all the variables are fresh, and let $\langle r_{id}, \langle w_rule \rangle \rangle$ an associated entry in R. Being ev be an event in Ev, and $\mathsf{triggering}(ev,R)$ a function returning $\langle r_{id}, \langle w_rule \rangle \rangle$ if exist in R an entry unifying the event ev, and $\perp$ otherwise, then the successful execution of w_rule will elicit the following transition in the artifact:

$$\frac{ev \in Ev \quad \mathsf{triggering}(ev,R) = \langle r_{id}, \langle w_rule \rangle \rangle \quad \mathsf{dispose}(ar_{id}) \notin w_rule_{body}}{\langle ar_{id}, ar_t, I, P, V, O \rangle \longrightarrow \langle ar_{id}, ar_t, I, P', V', O \rangle}$$

The artifact observable properties P may change according to the execution of the body of the triggered rules. In particular the transition may elicit the update to from P to P', where:

- $P' = P \cup \langle p_n, p_v \rangle$ for each property p_n added due to the execution of the body operators.

- $P' = P \setminus \langle p_n, p_v \rangle$ for each property p_n removed due to the execution of the body operators.
- $P' = P \setminus \langle p_n, p_v \rangle \cup \langle p_n, p'_v \rangle$ for each property p_n updated due to the execution of the body operators.

As in the execution of operation steps described in Section 5.3, the transition may also affect the artifact interface I and the artifact internal variables V. These changes are due to the particular specification provided by the artifact program art_{pr}.
Remarks: **(a)** As said, the body of a triggered workspace rule is executed atomically, no pending operations are allowed at the end of the transition. As a consequence, no op_ongoing events can be present in Ev at the end of the transition elicited by a workspace ruleand the artifact set O remains unchanged. **(b)** The execution of a dispose(ar_{id}) operator in the body of the triggered rule simply removes the artifact ar_{id} from the workspace. In this case the transition is not defined for the artifact ar_{id}.

Workspace Rules on Agents

Agents are affected by the execution of a workspace rule if they are focusing the involved artifacts according to the actual state of the observability map $Om \in M$. The effects of the transition elicited by a triggered entry in R are in updating the set of events collected by agents:

$$ag_{Ev} \longrightarrow ag'_{Ev}$$

The new event set ag'_{Ev} can be described by the summation of the effects that each operator pushes within the environment. Each operator applied by a workspace rule elicits a series of events which are described on the same basis of the transitions elicited by the operations described in Chapter 5.

6.5 Workspace Programming Examples

In order to give a concrete example of programming workspaces with workspace rules, in this section some simple applications are presented. In what follows, a simple composition of a counter infrastructure and a modified version of the producers consumers scenario are described.

```
class Counter1 extends Artifact
{
  ....
  @OPERATION void inc(){
    int c =
      getObsProperty("count").intValue();
    int newc = c + 1;
    if((newc % 2 ) == 0)
          triggerLinkedOp("C_e", "linkedInc");
    else
          triggerLinkedOp("C_o", "linkedInc");
    updateObsProperty("count",
          newc);
  }
}
```

```
class Counter2 extends Artifact
{
  ....
  @LINK void linkedInc(){
    int c =
      getObsProperty("count").intValue();
    int newc = c + 1;
    updateObsProperty("count", newc);
  }
}
```

Figure 6.2: *(left)* `Counter1` artifact type is used to maintain a global count used by agents. The logic to link an external operation has to be included inside the `inc` operation. *(right)* `Counter2` artifact type has to include a link operation in order to be linked by `Counter1` and count even or odd numbers only.

6.5.1 A Counter Infrastructure

Imagine to instrument a workspace with a simple counter infrastructure aimed at providing, besides a simple counter exploitable by agents, also a couple of observable properties indicating how many even and odd numbers have been counted so far. A first design choice would be to build a centralized artifact displaying three different observable properties, namely general count, even count and odd count. Whether we want a decentralised infrastructure, we may use the possibility to link the artifacts together, so to have a first counter (`C_a`) maintaining the global count and feeding two additional counters (`C_e` and `C_o`) displaying respectively the even and odd count properties. In this case, the adoption of the link operation may lead the programmer to use two different artifact types, as `Counter1` functioning as a master, and `Counter2` functioning as a slave. An excerpts of the CArtAgO implementation for `Counter1` and `Counter2` is showed in Figure 6.2. In particular, within the `inc` operation provided by `Counter1`, two link operations must be triggered on `C_e` or `C_o` (whether the global count is respectively even or odd). Then, a specific `Counter2` type has to be programmed to serve link operations, so to be triggered by the counter master.

The same problem can be suitably handled with the sole counter type defined in Subsection 4.3.1 by using workspace rules. In doing so, a couple of simple reactions can be specified to deviate events related to the master property update. Those events elicit the application of an `apply` operator triggering the `Inc` operation – respectively on `C_e` and `C_o` – as it follows:

```
+prop_updated("C_a", count, N)     +prop_updated("C_a", count, N)
:  N mod 2 == 0                    :  true
-> apply("C_e", Inc).              -> apply("C_o", Inc).
```

Besides the extreme simplicity of this solution, the improved separation of concerns, and the possibility to straightforwardly change the functioning of the whole infrastructure by simply changing the related workspace rules, the main convenience is, in this case, the possibility to fully reuse basic artifacts already available, without intervening on their programs.

6.5.2 Producers Consumers

As a second application showing the benefits of workspace rules we describe a revised version of the producers consumers scenario, which assumes to coordinate the activities of a set of producer agents providing some information item which has to be processed by a set of consumer agents. As described in Subsection 4.5.1, a Bounded Inventory artifact can be deployed in a workspace in order to mediate the interaction between agents. Introducing a Bounded Inventory artifact results in promoting coordination and synchronization of agents activities. In this case we aim at augmenting the environment support by providing an adaptive infrastructure to agents, namely an artifact based infrastructure which can be adapted according to special rules triggered by events occurring in the workspace. To this aim, an additional counter artifact is deployed, with the aim to count the total number of agents actually present in the workspace. As seen in Subsection 4.4.2, the counter artifact has a usage interface operation `inc`. We here introduce a second operation in the counter Ui, that is a `dec` operation to be used to decrement the count value. Given the described functioning, a couple of workspace rule entries can be specified at the workspace level in order to reckon the actual amount of agents actually present in the workspace and react accordingly. These rules are triggered every time agents join or leave the workspace:

```
+ws_joined(AG_id)                  +ws_leaved(AG_id)
: true                             : true
-> applyOp("Counter", inc).        -> applyOp("Counter", dec).
```

The body of the former rule contains an operator that simply triggers the execution of a `inc` operation on the counter. Besides, the reaction applied by the second rule elicits the execution of a `dec` operation on the counter. This makes it possi-

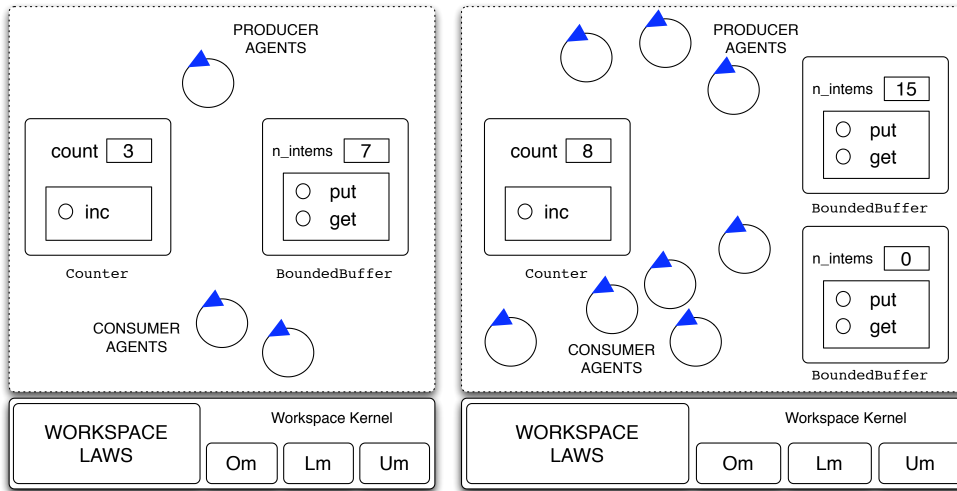

Figure 6.3: (*Left*) Producers Consumers modeled in artifact based workspace programmed with workspace rules. (*Right*) Once the number of agents reach a given threshold a workspace rule is triggered, and a new buffer artifact is automatically created in order to balance the computational load required for item exchanges.

ble to maintain an updated value of the amount of agents actually present in the workspace, while this value is also displayed by the `count` observable property.

A second workspace rule provides reactions every time the count reaches a multiple of N, where N is a fixed threshold (8 in this case). This control is entailed by the context condition specified by the rule, that in turns checks wether the actual amount of agents is multiple of N or not. The reaction in this case is applied for creating a new instance of Bounded Inventory and adopts a `make` operator:

```
+prop_updated("Counter", count, N)
    :  N mod 8 == 0
    -> make(BB_id, BoundedBuffer).
```

Such a simple program helps in balancing the computational load needed for item exchange: creating a new Bounded Inventory every time the overall group of agents augment of a given amount allows the system to maintain a fixed ratio between the number of agents and the resources (artifacts in this case) exploitable to fulfill the task.

6.6 Final Remarks on Intra-Workspace Dynamics

Programming intra-workspace dynamics involves multiple entities of the system, in so doing providing a mechanism of "crosscutting concerns" that may resemble, at an agentive level of abstraction, the one proposed in object oriented approaches by Aspect Oriented Programming (AOP) [75]. As aspects in AOP are assumed to alter the behavior of the base code (the non-aspect part of a program) by applying advices (additional behavior) at various join points, workspace rules are assumed to alter and transform the basic dynamics inside artifact based work environments. As AOP allows to modularise particular aspects of a complex program, where crosscutting functions are spread over a number of unrelated computational objects and components, workspace rules make possible to relate particular sequences of events scattered inside the workspace and originating since multiple agent activities. As AOP allows to prevent mixing several auxiliary aspects tangled with the basic functionality for which a component has been built for (i.e., its business logic concern), workspace rules allows to glue together events and changes occurring in different artifacts which initially have not been conceived to work together. For what concerns the scope of this work we here only remark that both AOP and workspace rules seems to fit the same family of programming issues. A most detailed comparison between the two approaches is behind our target challenges, and is let to future research.

As far as artifacts were introduced in MAS research, their main concern was about providing coordinating functionalities for easing agent activities [100]. Indeed, being entities shared and concurrently used by agents, artifacts are supposed to promote coordination between agents. Thereby, they can be straightforwardly deployed to instrument work environments as part of a *coordination media*, namely functioning as *coordination artifacts*. The use of coordination media is particularly relevant in all such contexts or problems where it is useful to adopt an *objective* approach to coordination as opposed to *subjective* approach to coordination [95, 115]. Whereas in subjective coordination approaches the coordination aspects are in charge of agents, typically objective coordination is reached by encapsulating the states and the rules defining the coordination policies in some proper controllable medium, that is placed as part of the system, out of the "minds" of agents. Objective coordination is particularly useful when: *(i)* the coordination laws are stable and the objective is to automate the coordination process as much as possible, thus minimizing the costs for coordination; *(ii)* There is no need of negotiation among the participants, which are even not required to know or understand the overall coordination strategy; *(iii)* the coordination rules

must be enforced besides the individual behavior of the participants (prescriptive coordination), but without violating their autonomy (i.e. control of their behavior). In our case, this is achieved by designing proper coordination tools as artifacts that agents create, share and use at runtime.

Workspace Rules can be seen as a further step in this direction as their adoption can be justified by the need to explicitly manage global coordination laws at the workspace level. In this view, programming workspaces with workspace rules provides a new degree of freedom for developers, defining a "locus" where to encapsulate global coordination mechanisms, which effects span from the single artifact functioning and reaches the boundaries of a workspace, viewed as the logical container of an application based on agents and artifacts. The coordination media approach can also be adopted to face those evolutionary and unpredictable aspects of open and complex systems (among others, this approach is followed in Casadei [21]). In these contexts, the overall coordination policies need to be changed at runtime, possibly without requiring to re-program from scratch the entities actually dwelling the work environment. Besides agents' capability to replace artifacts at runtime, or to inspect and possibly change/adapt artifact functioning, workspace rules provide an additional degree of control, exploitable either by agents and by developers to adapt on the need the rules governing the workspace.

This chapter concludes the part of the thesis addressed at programming environments in agent systems. The same aprroach who has lead at the definition of environmental infrastructures is pushed in the next part of the thesis, where an explicit organizational perspective will be taken in order to to realize objective patterns of coordination among agents.

Part III

Developing Organizational Infrastructures based on Artifacts

Chapter 7

Programming Organizations in Practice

This part of the thesis adopt an organization programming perspective to define those infrastructures inside the MAS to be exploited by agents for organizational purposes.

This chapter in particular envisages a programming model defining an organizational entity to be deployed as a concrete framework which agents can interact to in order to exploit organizational services. On the basis of the well suited $\mathcal{M}$oise organizational modeling language, a concrete example of organization will be sketched, according to a specification given along different conceptual dimensions. According to the specification of a concrete use case the programming model for a concrete organizational entity is provided, including the normative programming language addressed at specifying the overall organizational dynamics.

7.1 Taking the Organization Programming Perspective

On the basis of an organizational specification given along different conceptual dimensions, this chapter envisages a design model aimed at building concrete organizational entities to be deployed inside the MAS. The proposed model aims at introducing the basis for a concrete infrastructure which agents can interact in order to exploit organizational services. The approach taken in this chapter to specify an organizational entity adopts a methodology and a related technology

based on the Moise model. It follows and extends a research line that, in recent works by Hübner et al., leads to de definition of Organization Management Infrastructures [67, 63, 61, 62].

As introduced in Subsection 2.3.1, the specification of an organizational entity according the Moise approach is conceived in two phases. In a first phase the organization is designed by using an OML representation specifying in abstract terms the different dimensions and the relationships between the involved entities. In a second phase, such an abstract specification is translated to a norm based language (NPL) which in short can be fed to a the components capable to interpret and manage norms at runtime. Following the same approach, in the first part of this chapter the Moise Organization Modeling Language (OML) - already described in abstract terms in Chapter 2 - is adopted to define an abstract organizational specification in the context of a concrete case study. Moise is envisaged to collect and express specific constraints and cooperation patterns that the system developer (or the agents themselves) have in mind, thus resulting in an explicit program that can be referred as the organization specification (OS). The realization of a Moise OML specification is discussed in Section 7.2, taking the reference scenario as guideline.

The second part of the chapter (Section 7.3) describes the mechanisms originating an equivalent organizational specification, translated from the Moise OML and based on norms and institutional facts. Such a normative specification will be adopted to directly manage organizational entities, once they are deployed inside the system.

Finally, Section 7.4 concludes the chapter by discussing its contributes in the context of the rest of work.

7.2 Using Moise for modeling a concrete Organizational Entity

The Moise is a suitable programming model aimed at unifying different aspects underlying an organizational entity into an unified Organizational Modeling Language (OML) [65, 64]. As introduced in Section 2.3, the model provides a separate scope for each dimension, providing a unifying approach supporting the specification of the whole organization. In particular, Moise is characterized by the complete independence between the first two dimensions (functional and structural ones), which are related to the third one (the deontic dimension) by mean of

norms.

Throughout this section the example of an hospital surgery scenario will be used to illustrate and clarify the Moise-OML. The scenario resembles an ambulatory room as an open system, where heterogenous agents can enter and leave in order to fulfill their purposes. In particular, two types of agents are modeled as organization participants. *Staff agents* (namely physicians and medical nurses) are assumed to cooperate each other in order to provide medical assistance to visitors. Accordingly, *visitor agents* (namely patients and escorts) are assumed to interact themselves in order to book and exploit the medical examinations provided by the medical staff.

Taking hospital surgery scenario as a test case, the next sections give a brief description on how the three dimensions can be specified using Moise. In particular, the Moise graphical notation will be described, while Appendix A includes an equivalent specification in XML format. A complete description of programming organizations in Moise is outside the scope of this work, the interested reader can find more examples and application cases in [65, 64, 60] or in the project web page [84].

7.2.1 Structural Specification

The Moise Structural Specification (SS) provides the structure for the organization in terms of groups of agents, roles that can be played by agents participating in that group, and links defining functional relations between roles. The structural specification applied to the hospital scenario is showed in Figure 7.1 using the Moise graphical notation. It is built in three levels: *(i)* the behaviors that an agent is responsible for when it adopts a role (individual level), *(ii)* the acquaintance, communication, and authority links between roles (social level), and *(iii)* the aggregation of roles in groups (collective level).

The individual level defines roles that can be played by agents participating the organization. A role defines the scope for the behavior of agents actually playing it, thus providing a standardized pattern of behavior for the single individuals. An inheritance relation among roles can be specified, namely indicating roles that extends and inherits properties from parent roles. Figure 7.1 presents the Moise graphical notation for the SS in the context of the hospital scenario. As showed, a taxonomy of roles is provided. Visitor agents can adopt two roles, patient and escort, both inheriting from a visitor abstract role. Besides, two roles are defined since the figure of staff agents. The doctor is the role played by the physician inside the organization. It extends the properties of a more generic staff role,

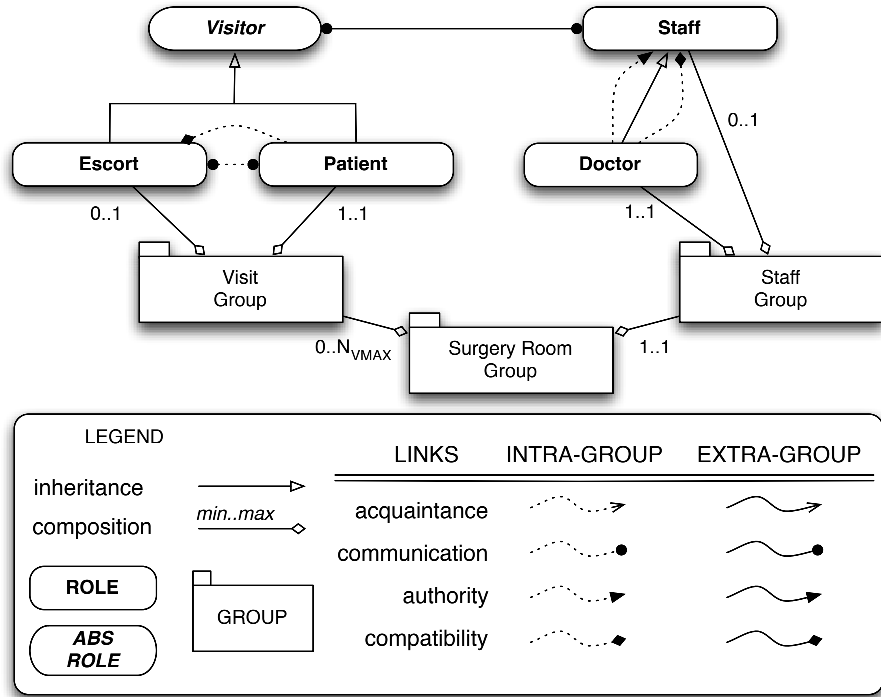

Figure 7.1: Structural Specification for an hospital surgery room using $\mathcal{M}$oise.

which is assigned in support and administration activities inside the team. From an agent perspective, the adoption of a role is constrained by a compatibility relation between roles. Hence, an agent can play two or more roles only if the roles to play are compatible. In the hospital scenario, the patient role is compatible with the escort role, while the doctor role is compatible with the staff role.

In the social level, a link relation is defined between roles, where each link relates a source to a target role. An authority link specifies that the source role has authority on the target role, as for instance the agent playing the doctor role has authority on an agent playing the staff role. Notice that in $\mathcal{M}$oise graphical representation the direction of the arrows unambiguously defines which are the target and the source of a link relation (see Figure 7.1). Besides, communication and acquaintance links are defined. Communication links allows two agents to communicate if they play roles with a communication link (as for instance between patient and escort, and between staff and doctor). The acquaintance link is similar. In order to simplify the specification, it can be assumed that for every au-

thority link there is an implicit communication link and for every communication link there is an implicit acquaintance link.

In the collective level, the participant agents are divided into groups, which specification consists in a set of roles and related properties and links. It is possible to have taxonomies of groups, namely it is possible to specify a set of inner-groups inside a group. The SS for the hospital scenario groups together escorts and patients (to form visit groups) and staff and doctor (to from the staff group). The specification allows intra-group links stating that an agent playing the source role is linked to all agents playing the target role, in spite of which groups these agents belong to. In the hospital SS a bidirectional intra-group link is specified, namely an agent playing the visitor role in the visit group is linked to all agents playing the staff role in the staff group, and viceversa. The composition of both the groups types forms the surgery room group, including the overall set of agents participating the organization. Notice that the cardinalities for agents playing a specific role inside a group are further specified. In so doing, zero or one single staff position can be present at the same time inside a staff group (cardinality range $[0..1]$) and exactly one position must be covered for the doctor role (cardinality range $[1..1]$). Besides, zero or one single escort position can be present at the same time inside a visitor group (cardinality range $[0...1]$) and exactly one position must be covered for the patient role (cardinality range $[1..1]$). In addition, the cardinality of groups inside composite groups can be specified. In the hospital scenario, the global surgery room group may include zero or more visit groups (cardinality range $[0..N_{VMAX}]$) and exactly one staff group (cardinality range $[1..1]$).

A well formed property can be ascribed to a given group once all the constraints imposed by the specification are respected. For instance, a well formed group is a group that *i* respect the compatibility of roles as it has been defined by compatibility links; *ii* respect the range of admissible agents actually playing each role, as it has been defined by role cardinality; *iii* respect the amount of inner-groups, as it as been defined by group; *iv* respect the number of participant agents as it has been constrained by a range indicating the minimum and the maximum number of participant; etc.
To resume, a group specified according to the SS can be defined formally as follows.

Definition 1 *(group)* According to the Structural Specification (SS), a group inside a $\mathcal{M}$oise Organizational Entity is defined by the elements of the following tuple:

$$\langle id, \mathcal{R}, \sqsubset, compat, maxrp, minrp, \mathcal{L} \rangle$$

where:

- *id* is an unique identifier for the group;
- $\mathcal{R}$ is a set of identifiers for roles that can be played in the group;
- $\sqsubset$ is a inheritance relation among roles $\in \mathcal{R}$;
- $compat : \mathcal{R} \rightarrow 2^{\mathcal{R}}$ is a function that maps each role to the set of its compatible roles;
- $maxrp : \mathcal{R} \rightarrow \mathbb{Z}$: is a function that maps each role to the maximum number of players of that role in the group (upper bound of role cardinality);
- $minrp : \mathcal{R} \rightarrow \mathbb{Z}$: is a function that maps each role to the minimum number of players of that role necessary for the group to be considered well-formed (lower bound of role cardinality);
- $\mathcal{L}$ is a set of links between roles in the scope of the group being defined;

The previous definition includes the basic elements defining a group inside the OML, namely the structures involved in its representation.

7.2.2 Functional Specification

The Functional Specification (FS) is composed by a set of functional schemes which describe how, in accord with the SS, various groups of agents are expected to achieve their global (organizational) goals [22]. In particular, FS specifies how organizational goals are decomposed by plans and how these plans are allocated to the individual agents. The related schemes[1] can be seen as goal decomposition trees, where the root is a goal to be achieved by the overall group and the leafs are goals that can be achieved by individuals, i.e. the single agents, through missions to be committed.

A goal decomposition can be set either by the MAS designers who specify their expertise in the scheme, but also by agents themselves, that could store their best practices[2].

[1] In the following, we use the term scheme for brevity in order to refer to the functional scheme.

[2] The capability to modify the structures of the organization – as uploading or replacing a scheme in the functional specification – requires agents that are aware of the organizational representations and functioning. In addition, they must have the capability needed to build a well formed scheme. In Section 2.3 we refer to these agents as *organizational agents*.

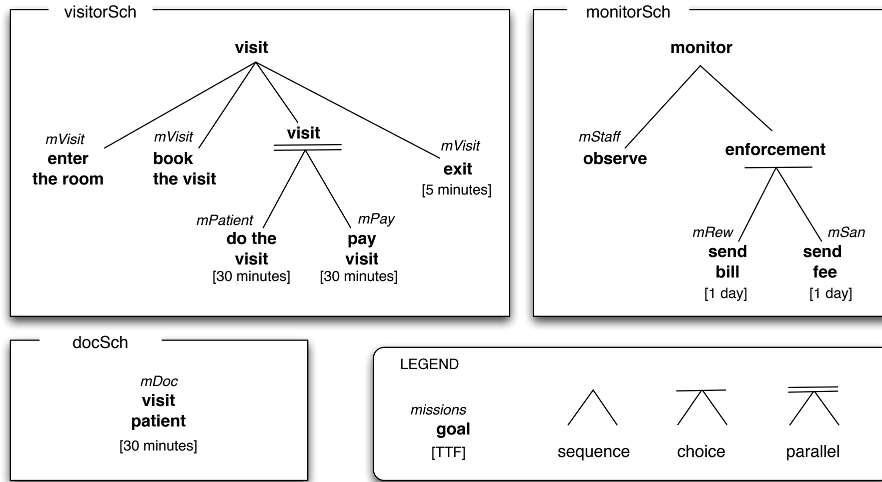

Figure 7.2: Moise functional schemes for agents working inside hospital surgery room.

In a scheme related to the functional specification, each non-leaf goal $g_j \in G$ (where G is the set of global goals) can be decomposed in sub-goals through plans. The following three operators are used to specify the decompositions:

sequence ",": the plan "$g_1 = g_2, g_3$" means that the goal g_1 will be achieved if and only if the goal g_2 and subsequently goal g_3 are achieved;

choice "|": the plan "$g_1 = g_2 | g_3$" means that the goal $g1$ will be achieved if one, and only one of, the goals $g2$ or $g3$ is achieved;

parallelism "‖": the plan "$g_1 = g_2 \parallel g_3$" means that the goal g_1 will be achieved if both g_2 and g_3 are achieved, but they can be achieved in parallel.

Each goal specified in the decomposition is related to a mission. A mission defines all the goals an agent commits to when participating in the execution of a scheme and, accordingly, groups together coherent goals which are assigned to a role in a group. According to the FS, an agent playing a given role can commit to the related mission, and thus bring about the included goals. More precisely, if an agent commits to the mission m_i, it commits to all goals of m_i ($\forall g_j \in m_i$), while the organization expects it will achieve them. Goals without an assigned mission are considered as fulfilled by the achievement of their sub-goals.

The analogy of an hospital surgery scenario is kept to clarify the model in Figure 7.2, where the functional model is specified using the $\mathcal{M}$oise graphical notation. In this case, the organization provides agents with three rehearsed schemes, through which the missions to be committed by agents are specified. The visitor scheme (`visitorSch`) describes the goal tree related to the visitor group. It specifies three missions, namely *mVisit* as the mission to which each agent joining the visit group has to commit, *mPatient* as the mission to be committed by the patient who has to undergo the medical visit, and *mPay* as the mission to be committed by at least one agent in the visit group. According to the goal decomposition tree set by the FS, the goals "do the visit" (which is related to the mission *mPatient*) and "pay visit" (which is related to the mission *mPay*) can be fulfilled in parallel, namely nothing forbids two different agents to pursue these goals at the same time. Besides, the *monitorSch* describes the activities performed by a staff agent. These plans are aimed at verifying if the activities performed by the visitors follows an expected outcome, namely if the visitors fulfill the payment committing the *mPay* mission (which includes the "pay visit" goal). The role staff is thus played by agents acting on behalf of the organization, namely agents which are assumed to detect norm violations and react accordingly[3]. Along its committed missions, a staff agents is thus concurrently observing the visitor activities ("observe" goal). In this case, the sub goals related to the "enforcement" goal are mutually exclusive, namely only one has to be fulfilled and, of course, they cannot be pursued in parallel. Enforcing goals are aimed at promoting a conforming behavior to the visitor agents, which in turns can be sanctioned ("send fee" goal) or rewarded ("send bill" goal, the reward being a discount). Finally, the *docSch* specifies the activities to which a doctor has to commit, namely to perform the visit to every patient. Notice that each mission has a further property specified between square brackets in the FS: it refers to the maximum amount of time the organization expects for the agent to commit to the mission ("time to fulfill", or ttf value). For instance, according to the *docSch* given in the hospital scenario, a doctor has 30 minutes to commit the mission *mDoc*. Notice also that, along with the goal decomposition, a scheme also specifies the pre-condition for a given goal g_j, namely goals that must be fulfilled in order to achieve the following goal g_j of the scheme. In so doing, an agent who committed to the mission m_i which includes the goal g_j, is expected to fulfill the g_j only when g_j preconditions are satisfied. For instance, a visitor can achieve the goal "book the visit" only when

[3]The particular role in the context of the organizational entity can be associated to the notion of organizational agent, as it has been described in Section 2.3.

mission	*cardinality*
mVisit	1..2
mPatient	1..1
mPay	1..1
mStaff	1..1
mSan	1..1
mRew	1..1
mDoc	1..1

Table 7.1: Mission cardinalities for the hospital scenario specified for the schemes defined in $\mathcal{M}$oise FS (Figure 7.2).

the goal "enter the room" is already satisfied; a patient may "do the visit" only if someone has already achieved the goal to "book the visit", and so on.

The FS also defines the expected cardinality for every mission in the scheme, namely the number of agents inside the group who may commit a given mission without violating the scheme constraints. Table 7.1 shows the mission cardinalities in the context of the hospital scenario, where the range is specified through the upper and the lower bounds for admissible commitments. In this case every provided mission has to be committed by exactly one agent except the *mVisit* mission, that has to be committed by all the agent in the visit group.
To resume, a functional scheme can be defined formally as follows.

Definition 2 *(scheme)* According to the Functional Specification (FS), a scheme is represented inside the $\mathcal{M}$oise Organizational Entity by the following tuple:

$$\langle id, \mathcal{M}, maxmp, minmp, \mathcal{G}, gm, gpc, ttf, rg \rangle$$

where:

- *id* is a unique identifier for the scheme;
- $\mathcal{M}$ is a set of identifiers of missions that agents can commit to in the context of the scheme;
- $maxmp : \mathcal{M} \rightarrow \mathbb{Z}$ is a function that maps each mission to the maximum number of commitments of that mission in the scheme (upper bound for mission cardinality);

- $minmp : \mathcal{M} \rightarrow \mathbb{Z}$: maps each mission to the minimum number of commitments of that mission necessary for the scheme to be considered well-formed (lower bound of mission cardinality);
- $\mathcal{G}$ is a set of goals of the scheme;
- $gm : \mathcal{G} \rightarrow \mathcal{M}$: maps goals to the related missions;
- $gpc : \mathcal{G} \rightarrow 2^{\mathcal{G}}$: maps goals to their pre-condition goals;
- $ttf : \mathcal{G} \rightarrow \mathbb{Z}$ maps goals to their "time to fulfill";
- $rg \in \mathcal{G}$ is the root-goal of the scheme.

The previous definition includes the basic elements defining a scheme inside the OML, namely the structures involved in its representation.

7.2.3 Deontic Specification

The deontic dimension describes explicitly what are the rights and the duties for the agents in order to respect the rules and the constraints specified by the organization. In this view, the deontic model glues all the other dimensions in a coherent organization. Indeed, the corresponding Normative Specification (NS) relates roles (as they are specified in the SS) to missions (as they are specified in the FS) by specifying a set of norms.

The $\mathcal{M}$oise approach to norms assumes that each norm is placed in terms of *permissions* or *obligations* to commit to a mission. This allows goals to be indirectly related to roles, i.e. through the policies specified for mission commitment (as described in Subsection 7.2.2, a mission for a single agent can be retrieved since a goal decomposition tree)[4]. In abstract terms, a $permission(\rho, m, ttf)$ states that an agent playing the role ρ is allowed to commit to the mission m within the deadline ttf. Similarly, $obligation(\rho, m, ttf)$ states that an agent playing ρ is obliged to commit to m within the deadline ttf. On the other hand, $\mathcal{M}$oise assumes *prohibitions* "by default" with respect to the specified missions: namely,

[4]An important feature of $\mathcal{M}$oise is to avoid the direct link between roles and goals. One reason is to define sets of coherent goals (the missions) which are not reducible to the concept of role. In so doing, roles are indirectly linked to missions by means of norms, namely permissions and obligations. Another reason is to improve the independence between the functional and the structural dimensions.

id	*condition*	*role*	*type*	*mission*	*TTF*
$n1$		$Escort$	*obligation*	$mVisit$	–
$n2$		$Patient$	*obligation*	$mVisit$	–
$n3$		$Patient$	*obligation*	$mPatient$	–
$n4$		$Escort$	*permission*	$mPay$	5 minutes
$n5$	$unfulfilled(n4)$	$Patient$	*obligation*	$mPay$	5 minutes
$n6$		$Staff$	*obligation*	$mStaff$	–
$n7$		$Doctor$	*obligation*	$mDoc$	–
$n8$	$unfulfilled(n5) \wedge unfulfilled(n4)$	$Staff$	*obligation*	$mSan$	1 day
$n9$	$fulfilled(n4) \vee fulfilled(n5)$	$Staff$	*obligation*	$mRew$	1 day
$n10$	$unfulfilled(n6)$	$Doctor$	*obligation*	$mStaff$	–

Table 7.2: Deontic description in Moise specifying norms regulating the hospital surgery room scenario.

if the normative specification does not include a permission or an obligation for a role-mission pair, it is assumed that the role does not grant the right to commit to the mission. Deadlines, namely "time to fulfill" (ttf) values, refer to the maximum amount of time the organization expects for the agent to fulfill a norm. This additional constraint has been included in Moise^{INST} [10] in order to bind agent behavior in a timely fashion.

Each norm is specified in Moise by its identifier *id*, by its triggering condition c, by its role ρ, by its type t (obligation / permission), by a related mission and a time to fulfill (ttf) value[5]. Given the above specification, each norm provides the constraints for the agent to commit to the missions specified for their adopted role. Table 7.2 shows the normative specification for the hospital scenario, and refers to the missions described in Figure 7.2. For instance, norms $n1$ and $n2$ define an obligation for agents playing either patient and escort roles to commit to the *mVisit* mission. A patient is further obliged to commit to *mPatient* mission ($n3$). Similar norms are provided in $n6$ and $n7$, where agents playing staff and doctor roles are obliged to commit to *mDoc* and *mStaff* respectively. The norm $n10$ is activated only when the norm $n6$ is not fulfilled (the deadline for $n6$ is not set in this case, thus its default value is 0, which means "now"). It specifies an obligation

[5] The values for the norm condition and the time to fulfill are optional in the normative specification: once not provided their default value are respectively set to $true$ and 0.

for a doctor to commit the *mStaff* mission, if no other staff agent is committing to it inside the group. *n*4 allows to agents playing as escorts the permission to commit to the mission *mPay*. Wether this norm is not fulfilled before the deadline $ttf = 5$ minutes, the norm *n*5 is instantiated, obliging the patient to commit the same mission *mPay*. This allows a double control upon the payment goal, which in turns has to be fulfilled by at least one agent inside the visit group.

The FS also allows to specify meta-norms, namely norms triggered by events indicating the violation of other norms. This allows to tackle a possible norm violation by the mean of further norm activations. The organization assumes the presence of organizational agents, namely staff agents in this case, which are assumed to detect norm violations and react accordingly. Along its committed missions, a staff agents is concurrently observing the visitor activities (agents playing the role staff are assumed to fulfill the monitor scheme, or *monitorSch* defined in Figure 7.2). These agents are assumed to detect norm violations and suddenly apply sanctioning policies. For this purpose, norms *n*8 and *n*9 are specified. In norm *n*8, whether both norms *n*4 and *n*5 are unfulfilled after the deadline (5 minutes), a staff agent is obliged to commit to the *mSan* mission, which goal in short is assumed to sanction the patient. Besides, in norm *n*9, whether either *n*4 or *n*5 are fulfilled before the deadline (5 minutes), a staff agent is obliged to commit to the *mRew* mission, which goal in short is assumed to reward the patient.
To resume, a norm in $\mathcal{M}$oise can be defined formally as follows.

Definition 3 *(norm)* A norm in the Deontic Specification (NS) is represented inside the Organizational Entity by the following tuple:

$$\langle id, c, \rho, t, m, ttf \rangle$$

where:

- *id* is a unique identifier for the norm;
- *c* is the activation condition for the norm;
- ρ is the role;
- *t* is the type (obligation or permission);
- *m* is the mission;
- *ttf* indicates “time to fulfill”, namely the deadline within which the norm has to be fulfilled.

```
      ⟨np⟩  =  "np" ATOM "{" {⟨np_fact⟩|⟨np_rule⟩|⟨np_norm⟩} "}"
 ⟨np_fact⟩  =  ATOM"."
 ⟨np_rule⟩  =  ATOM ":-" ⟨np_exp⟩"."
 ⟨np_norm⟩  =  "norm" ID":" ⟨np_exp⟩ "->" (⟨fail⟩|⟨obl⟩)"."
--------------------------------------------------------------------
    ⟨fail⟩  =  "fail(" ATOM ")."
     ⟨obl⟩  =  "obligation( "(VAR|ID) "," ATOM "," ⟨np_exp⟩
                "," ⟨time⟩ ")."
--------------------------------------------------------------------
  ⟨np_exp⟩  =  ATOM | "not" ⟨np_exp⟩ | ATOM ("&"|"|") ⟨np_exp⟩
    ⟨time⟩  =  "‘"( "now" | NUMBER ("second"|"minute"|...) )"’"
               [("+"|"-")⟨time⟩]
```

Table 7.3: Syntax of Normative Organization Programming Language - NOPL (adapted from [61]).

We can read that norm as "when c holds, the agents playing the role ρ are t to commit to mission m before ttf". It includes the basic elements defining a norm inside the OML, namely the structures involved in its representation.

7.3 From the Moise specification to a Normative Specification

As introduced in Subsection 2.3.3, NPL can be obtained by a translation of a OML specification into a language based on symbolic norms, rules and institutional facts. Normative Organization Programming Language (NOPL) is a particular class of NPL obtained once the OML (the source language) is provided on the basis of the Moise model. As presented by Hübner et al. in [61, 62], the translation mechanism from the Moise language to the target NOPL is based on a set of *translation rules* which are procedurally applied to the starting specification. It is worth to remark that, although the approach followed here is addressed to a Moise specification, other source OML could be translated to obtain different NPL, as different transformation procedures could be introduced for instance starting form AGR or AMELIE sources[6]. In this case, the NOPL language obtained after the application of the translation rules is the code used by the internal interpreter implemented inside the organizational entities deployed inside the MAS. Figure 7.3 shows the components involved it the management of a NOPL speci-

[6]Of course, starting from different OML would cause a different configuration for the adopted organizational entities.

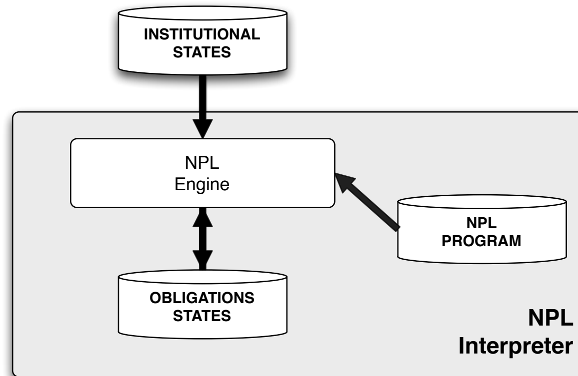

Figure 7.3: A NOPL (Normative Organizational Programming Language) specification can be automatically obtained by translating a Moise OML specification. NOPL stores normative rules, inference rules and organizational/institutional facts which are interpreted by an engine embedded inside the organizational entity, so to update at runtime its configuration.

fication inside an organizational entity in this case the NOPL program translates the normative specification of the organization.

The NOPL syntax is provided in Table 7.3 (given in [61]). It consists in a normative program where, besides norms (*np_norms* according to the syntax), it is possible to express constructs to store predicates as organizational states (*np_facts*) and also rules to operate over internal data structures (*np_rules*).

7.3.1 Normative Organization Programming Language

We here provide an example of Moise-OML to NOPL translation, based on the mechanism described in [62]. An excerpt of the resulting NOPL in the context of the hospital scenario is then discussed. As showed in Figure 7.3, the organizational entity is regulated by an internal NOPL engine referring to the organizational/institutional states given by the functional dimension of the organization, namely schemes and obligations[7]. To obtain a NOPL specification a set of *transla-*

[7] In this context, and in the following of this work, we refer to *organizational facts* and *institutional facts* as synonyms, indicating those structures and statements that represent the actual organizational state. We are aware that reducing to a common ground these notions would require a deep analysis on what related researches have defined for computational institutions as opposed

tion rules are applied to the $\mathcal{M}$oise specification. We first deal with the translation of the functional dimension as expressed in $\mathcal{M}$oise. As seen in Subsection 7.2.2, a scheme is represented by the following tuple:

$$\langle id, \mathcal{M}, maxmp, minmp, \mathcal{G}, gm, gpc, ttf, rg \rangle \in FS$$

The difference between the notion of time adopted here with respect to the one given within the workspace configuration has to be remarked. Although both adopt, of course, a linear and discrete notion of time, they in fact depend on two different underlying execution systems. The former time refers to the time provided by the organization, hence used at an application level, i.e. to regulate obligations lifecycle and eliciting potential transitions in the norm states. On the other hand, the latter time is used by the system running the work environment (i.e. CArtAgO), thus providing synchronization of interactions and sorting of general events inside the workspace.

NOPL Facts

The translation applied to the FS instance produces at runtime the following facts in NOPL:

- `scheme_mission`(m, min, max): is a fact that defines the cardinality of a mission $m \in \mathcal{M}$;
- `goal`($m, g, pre-cond, `ttf`$): is a fact that defines the arguments for a goal $g \in \mathcal{G}$: its mission m, the pre-conditions, and time to fulfill (deadline);
- `mission_role`(m, r): the role r is permitted or obliged to commit to mission m, according to the normative specification given by $\langle id, c, \rho, t, m, ttf \rangle \in NS$;

The following dynamic facts will be provided at runtime by the artifact that manages the scheme instance:

- `plays`(a, ρ, gr): agent a plays the role ρ in the group instance identified by gr.

to computational organizations. But, as in [132], we assume that the both the concepts behind institutions and organization oriented approaches reflect the same basic modeling principles. Both the approaces are in fact addressed at reifying organizational entities in MAS through a set of shared social concepts (norms, roles, etc.).

```
// number of players of a mission M in
// scheme S
// .count(X) is a function that counts
// how many instances of X are known

mplayers(M, S, V) :-
  V = .count(committed(_, M, S)).

// well-formed scheme S

well_formed(S) :-
  mplayers(m, S, Vm)
  & Vm >= minmp(m)
  & Vm <= maxmp(m).
```

```
// goal G of scheme S is ready:
// all its pre-conditions have been achieved

ready(S,G) :-
  goal(_, G, PCG, _) & all_achieved(S, PCG).

all_achieved(_, []).
all_achieved(S, [G|T]) :-
  achieved(S, G, _) & all_achieved(S, T).
```

Table 7.4: NOPL rules: (left) rules to infer a well formed scheme; (right) rules for establishing ready goals.

- `responsible`(gr, s): the group instance gr is responsible for the missions identified by the scheme instance s.
- `committed`(a, m, s): agent a is committed to mission m in scheme s.
- `achieved`(s, g, a): goal g in scheme s has been achieved by agent a.

NOPL Rules

Besides facts, a set of *np_rules* are created to infer the number of agents which are actually committed to a mission, the state of the scheme (e.g. whether it is well-formed or not) and the state of the goals (e.g. whether they are ready to be achieved or not). The related NOPL cutout is showed in Table 7.4. Notice that the `ready` and `well-formed` rules express the semantic of ready goals and well-formed schemes in terms of the $\mathcal{M}$oise model (Table 7.5, left). The well-formed rule, in particular, is specific for the scheme being translated and controls whether the number of players actually committed to a given mission m respect the cardinalities specified in the scheme. Notice that, once applied, the translation rules automatically generate the NOPL code by unifying the variables (represented in italic fonts in Table 7.5) with the related values actually stored in the NOPL facts.

```
// agents are obliged to fulfill
// their ready goals

norm ngoa:
  committed(A, M, S)
  & goal(M, G, _, D)
  &  well_formed(S)
  &  ready(S, G)
  -> obligation(A, ngoa,
      achieved(S, G, A), 'now' + D).
```

```
// goal fulfillment is regimented

norm goal_non_compliance:
obligation(Ag, ngoa(S,M,G), Obj, TTF)
& not Obj
& ''now'' > TTF
-> fail(goal_non_compliance(
    obligation(Ag, ngoa(S,M,G), Obj, TTF))).
```

Table 7.5: NOPL norms for goals. (Left) The norm `ngoa`, regulates goals fulfillment and provides the semantic for their commitment. (Right) Based on the norm `ngoa`, the norm `goal_non_compliace` regulates compliances to obliged goals.

```
norm mission_cardinality:
  scheme_mission(M, _, MMax) &
  mplayers(M, S, MP)
  & MP > MMax
  -> fail(mission_cardinality).
```

```
norm mission_cardinality:
  scheme_mission(M, _, MMax)
  & mplayers(M, S, MP) & MP > MMax
  & responsible(Gr, S) & plays(A, ρ, Gr)
  -> obligation(A, mission_cardinality,
        committed(A, m,  _), 'now' + ttf).
```

Table 7.6: NOPL program related to property norms: (left) the norm is regimented; (right) the norm is enforced through an obligation.

```
norm id:
  plays(A, ρ, Gr)
  & responsible(Gr, S)
  & mplayers(m, S, V)
  & V < maxmp(m)
  -> obligation(A, id,
       committed(A, m, S),
       'now'+ttf).
```

```
norm mission_permission1:
  committed(Ag, M, S)
  & plays(Ag, R, _)
  & not mission_role(M, R)
  -> fail(mission_permission(Ag, M, S)).

norm mission_permission2:
  committed(Ag, M, S)
  & plays(Ag, R, Gr)
  & mission_role(M, R)
  & responsible(Gr, S)
  -> fail(mission_permission(Ag, M, S)).
```

Table 7.7: NOPL program related to domain norms: (left) norms regulating mission commitment; (right) norms for prohibitions are expressed by regimentation.

NOPL Norms

The *np_norms* which are of concern for the scheme management can be of three types: norms for goals, norms for properties and domain norms.

NORMS FOR GOALS. The first class of norms refers to norms for regulating the fulfillment of organizational goals, and includes norms expressing the $\mathcal{M}$oise semantics for commitments. According to the norm `ngoa` (Table 7.5 left), an obligation is entailed for an agent `A` to achieve the goal `G` in the scheme `S` before the deadline `'now' + D`. This obligation is created when the agent `A`: *(i)* is committed to a mission `M` that *(ii)* includes a goal `G`, *(iii)* the mission scheme `S` is well-formed, and *(iv)* the goal is ready.

An important norm is related to the previous one in order to regulate obligations to achieve goals. Thereby, according to the `goal_non_compliance` norm, a fail event is generated when: *(i)* an agent `Ag` is committed to the mission `M` in the scheme `S` related to the group `G`, *(ii)* the agent is committed to fulfill `Obj`, and *(iii)* the deadline `TTF` set to fulfill `Obj` is passed.

NORMS FOR PROPERTIES. The second class of norms refers to properties guaranteeing the global coherence of the organization, as they can be specified in the $\mathcal{M}$oise model. An example of this type of norms can be related to the $\mathcal{M}$oise property specifying mission cardinality. As described in Subsection 7.2.3, a norm expressed in $\mathcal{M}$oise is represented by $\langle id, c, \rho, t, m, ttf \rangle \in NS$ In this case, the norm has to define the consequences for a circumstance where there are more agents committed to a mission than permitted by the scheme specification. The condition triggering the norm is $c = \#mc$, where $\#mc$ denotes the maximum cardinality allowed for that mission.

As seen in Subsection 2.3.3, to manage the norm violation two kinds of consequences are possible, namely obligation and regimentation. Regimentation is the default consequence in NOPL and it is used when there is no norm with condition $c = \#mc$ in the normative specification. Otherwise, the consequence will be an obligation. The organization developer may choose one or the other when specify the organization. The two options are translated in two different NOPL programs, as showed in Table 7.6. On the left, the norm regulating mission commitment is regimented, namely each tentative to commit a mission that has reached its maximum cardinality will fail. On the right, the norm creates an obligation for the agent to commit to the mission m before the deadline specified by ttf.

DOMAIN NORMS. The third class of norms translates the domain norms, namely the those obligations that have been specified also by the normative structures inside the organization specification. In this case norm is produced in NOPL since each norm expressed in $\mathcal{M}$oise. Whereas $\mathcal{M}$oise obligations refer to roles and missions, NOPL requires that obligations are for agents and towards a goal. A NOPL norm in this case identifies the agents playing the role in groups re-

```
scheme_mission(mVisit, 1, 2).
scheme_mission(mPatient, 1, 1).
scheme_mission(mPay, 1, 1).

...
goal(mPatient, do_the_visit,
   [book_the_visit], 'now' + 30 minutes).
goal(mPay, pay_visit,
   [book_the_visit], 'now' + 30 minutes).
goal(mVisit, exit,
   [do_the_visit], 'now' + 5 minutes).
...

mission_role(mVisit, patient).
mission_role(mVisit, escort).
mission_role(mPatient, patient).
```

```
mplayers(M, S, V) :-
  V = .count(committed(_, M, S)).


well_formed(S) :-
  mplayers(mVisit, S, VmVisit)
        & VmVisit >= 1 & VmVisit <= 2
  & mplayers(mPatient, S, VmPatient)
        & VmPatient >= 1 & VmPatient <= 1
  & mplayers(mPay, S, VmPay)
        & VmPay >= 1 & VmPay <= 1.
```

Table 7.8: Example of NOPL facts and rules for the hospital surgery scenario.

```
norm mission_cardinality:
  scheme_mission(M, _,  MMax)
  & mplayers(M, S, MP)
  & MP > MMax
  -> fail(mission_cardinality(M)).
```

```
norm role_cardinality:
  group_id(Gr) &
  group_role(R, _ ,RMax)
  & rplayers(R, Gr, V)
  & V > RMax
-> fail(role_cardinality(R)).
```

Table 7.9: Example of NOPL program regulating mission and role cardinality.

sponsible for the scheme and, if the number of players still does not reach the maximum, the agent is obliged to achieve a state where it is committed to the mission (Table 7.7, left).

The NS also specifies permissions and (by default) prohibitions. Since everything is permitted by default in NOPL, NS permissions do not need to be translated. Besides, each NS prohibition is handled in NOPL by the generic norms expressed in Table 7.7 (right). The first norm handles the case where the agent is committed to a mission that is not related to its roles (see the `mission_role` fact above). In the second case, even if the role and the mission are related, the agent does not play the corresponding role in groups responsible for the scheme.

7.3.2 NOPL in practice: the Hospital Scenario

In order to provide a concrete taste of the approach, in this section the NOPL program presented above is described in the context of the hospital surgery sce-

```
norm n1:
  plays(A, escort, Gr)
  & responsible(Gr, S)
  & mplayers(mVisit, S, V)
  & V <  2
  -> obligation(A, n1,
      committed(A, mVisit, S), 'now').

norm n2:
  plays(A, patient, Gr)
  & responsible(Gr, S)
  & mplayers(mVisit, S, V)
  & V <  2
  -> obligation(A, n2 ,
      committed(A, mVisit, S), 'now').
```

```
norm n6:
  plays(A, staff, Gr)
  & responsible(Gr, S)
  & mplayers(mStaff, S, V)
  & V <  1
  -> obligation(A, n6 ,
      committed(A, mStaff, S), 'now').

norm n7:
  plays(A, doctor, Gr)
  & responsible(Gr, S)
  & mplayers(mDoc, S, V)
  & V <  1
  -> obligation(A, n6 ,
      committed(A, mDoc, S), 'now').
```

Table 7.10: NOPL example regulating mission commitment.

nario introduced in Chapter 7. The NOPL code is automatically generated since the *translation rules* described in the previous section, where, in particular, the variables are replaced by the values actually dwelling the organizational states. Table 7.8 (left) shows an excerpts of NOPL program stating institutional facts translated from the $\mathcal{M}$oise specification[8]. Besides, Table 7.8 (right) shows the NOPL rules denoting the well-formed property for the *visitorScheme*. The code is produced on the basis of the the rule described in Table 7.5, where the variables are replaced by the states stored in the FS representation, as it has been given in Subsection 7.2.2.

Table 7.9 (left) shows an excerpts of NOPL program stating norms regulating mission cardinality. Since no norms have been specified in the NS to regulate a surplus of mission commitments (see Subsection 7.2.3), the default norm is applied, namely the commitment is regimented and no more agents will be able to commit to a mission that has reached its *maxmp*. The code is produced since the rule introduced in Table 7.6 (left). Similarly, Table 7.9 (right) shows the NOPL program expressing norms regulating role cardinality[9]. Notice that the applicability of these norms is checked against the context provided by an expression *np_exp*. Hence, in order to activate the norm, *np_exp* has to entail the facts actually stored in the NOPL program. In the example, in order to regiment the

[8] The complete NOPL specification related to the hospital scenario described here can be found in Appendix B.

[9] The rules to obtain NOPL facts since $\mathcal{M}$oise SS are omitted for brevity. These rules allow to specify facts related to the groups and the roles played inside the group, as for instance `group_role(R, _, M)` and `rplayers(R, Gr, V)`.

adoption of role R due to the `role_cardinality` norm, the number of players for the role R must be equal to the maximum cardinality specified in the SS for R.

Table 7.10 shows NOPL norms issued from translation of norms $n1, n2, n6$ and $n7$, as they have been specified in Moise FS described in Table 7.2. In this case, the specified norms relates functional schemes to roles, and indicate agent obligations for mission commitment.

7.4 Final remarks on Programming Organizations in Practice

Defining a use case in a specific application domain, this chapter proposed practical approach to design concrete organizational entities in MAS. As based on Moise OML [65], a series of different modeling dimensions are envisaged in order to cover the multifaceted requirements of complex organizations. They concern in particular structural, functional and deontic aspects, which are mobilized to define rich organizational patterns in order to control and promote coordination and cooperation in MAS. These levels are assumed to shape the overall organization in groups of agents, also defining roles that agents will play in the context of each group, as well as the missions related to agents ability to fulfill the expected tasks. An explicit treatment of norms allows to bind all the organizational dimensions together and, besides, to fix behavioral constraints to agents. Norms impose to agents obligations and permissions that can be either regimented or enforced by the organization at an application level. Finally, a mechanism for translating the Moise specification to a Normative Programming Language has been introduced, based on the mechanisms defined in [61, 62]. The normative language, based on the concrete specification of the use case, has been adopted as the program regulating the functioning of a concrete organizational entity to be deployed inside MAS.

On these basis, in the next chapter the presented modeling approach will be applied at a programming level, thus reifying the whole organizational entity as a set of organizational functionalities and services exploitable by agents at an application level. In shifting from the organizational specification to the organizational deployment, a main concern will be to not lose the level of abstraction adopted to model organizational entities. The practical aspects of implementing an organization inside the system will be addressed by shaping a so called Organization Management Infrastructure (OMI), in particular based on the A&A approach. Thereby,

the organization will be conceived as a distributed, decentralised infrastructure instrumenting the MAS work environment with a set of specialized artifacts. The adoption of artifacts will be a pivotal feature in the following of this work: finally, it will allow a seamless integration of Organizational Entities with other heterogeneous services instrumenting the MAS, thus providing a complete support for societies of agents engaged in their complex purposive activities.

Chapter 8

Organizational Management Infrastructures based on Artifacts

Once the general structures involved in a organizational entity have been analyzed, the problem is how to build an organizational entity in practice, and how to effectively deploy the organizational entity so as to be suitably exploited by agents in order to support either individual and collective activities. The approach adopted in this chapter is based on A&A model and makes use of artifacts as the unique abstraction tool to implement the organizational entity. Organizational artifacts are thus deployed in the work environment to instrument workspaces with organizational facilities, thus reifying and modularizing the functional-part of the organizational entity. The result is a decentralised and configurable layer of organizational resources, an organizational infrastructure, that can be perceived and used by agents as first-class entities of their work environment. Besides, the enabling and governing function inherited by artifacts allows the infrastructure to effectively enact norms inside the system. Through an internal normative model providing the infrastructures with a flexible mechanism to dynamically manage institutional states, norms are first monitored during their life cycle, and then applied according to regimentation or enforcement policies.

8.1 Shaping Organizational Management Infrastructures with A&A

The design model of an organization centered MAS typically follows a common trend, by introducing mechanisms responsible for providing agents with organiza-

tional services aimed at enabling and ruling their activities with normative control. As noticed in [63], a recurrent design model in organizational systems is the presence of middleware components (i.e., organizational proxies) that are assumed to mediate interactions between agents participating the organization and the infrastructures providing organizational functionalities[1]. On these basis, a series of modeling drawbacks rise when fine grained interactions between agents and organizations are of concern. A first major issue is the mismatch at the abstraction level between the entities playing the system. Whereas the organization is conceived as a series of services at a slightly "abstract" level, i.e. defined in terms of norms, roles, global goals, etc., agents are built upon mental attitudes as beliefs, desires, intentions (assuming BDI-like agents as the reference model). As a consequence, organization and agents are "detached" elements of the system. Organizational infrastructures should be viewed by agents either as bridge enabling the execution of external actions, either as the source of multiple perceptions, which filtering allows to control of their ongoing activities. This places the problem to bridge the conceptual gap between the elements and the resources available at the organizational level and the notions and the constructs adopted at the agent level. Finally, middleware based organizations bias an overwhelming power over agents which are typically simply participants, hence placing the issue to play an active role in managing the system, i.e. by creating, modifying, adapting the organization on the need.

A first attempt to overcome these issues has been proposed in ORA4MAS [63], an organizational entity which provides organizational functionalities shaped as an infrastructure based on a decentralised set of artifacts. In ORA4MAS artifacts are viewed as passive computational entities embedding organizational functions, which can be exploited by agent interacting with artifacts In order to improve the separation of concerns – either at a design level, either at a programming one – two kinds of agents are assumed to inhabit the organization: *(i)* participating agents are assumed to join the organization in order to exploit its services as users (i.e., adopting roles, committing missions etc.), while *(ii)* organizational agents are considered as organization aware agents, hence assumed to manage the organization by making changes to its functional and structural aspects (i.e., creating and updating functional schemes or groups) or to make decisions about the deontic events (i.e. norm violations). In ORA4MAS the presence of organizational agents is explicitly envisaged in order to manage the infrastructure (i.e. by cre-

[1] Such an approach is for instance followed by $\mathcal{S}$-$\mathcal{M}$oise$^+$ [67], MadKit [57], AMELIE [47] and almost all the models described in Chapter 2.

ating, deleting and replacing the involved artifacts). ORA4MAS infrastructure adopts four artifact types they are assumed to realize the whole organizational entity covering specific functions, being each artifact type specialized for a specific organizational dimension: *normative* artifact, *scheme* artifact, *group* artifact, and *org* artifacts. Using artifacts as basic building blocks of the OMI allows agents to natively interact with the organizational entity at a proper abstraction level, namely without being constrained to shape external actions as low level primitives needed to work with middleware objects and components as, for instance, in $\mathcal{S}$-$\mathcal{M}$oise$^+$. The consequence is that every interaction with the organizational entity can be shaped by an agent on native capabilities as actions and perceptions. Besides, the infrastructure does not rely on a sort of hidden organizational layer, but it is placed beside the agents as a suitable set of services and functionalities to be exploited as an integral part of the work environment. Moreover, differently form approaches based on middleware components, an OMI based on artifacts can be dynamically adapted and possibly replaced (by agents themselves) during the whole organization lifetime.

The roadmap to organizational infrastructures followed in this chapter relies on the conceptual model proposed by ORA4MAS and makes use of artifacts and workspaces as the main abstraction tools to implement the OMI as an infrastructure instrumenting the work environment. As in ORA4MAS, artifacts are used to instrument workspaces and to reify and modularize the functional-part of the organization machinery, as a distributed set of organizational resources and computational tools that can be perceived and used by agents as first-class entities of their work environment. Placed in terms of organizational artifacts (described in Section 8.2), the OMI "inherits" artifact computational model as it has been defined in Chapter 4, thus providing a suitable mean for agents who want to join and work inside an organization. Differently from ORA4MAS, and similarly to the recent proposal made by Hübner et al. in [61, 62], the organizational infrastructure proposed here includes the possibility to explicitly manage norms, as they are represented inside the organizational entity through a NOPL programming model, extract from the the $\mathcal{M}$oise specification.

The computational model adopted for programming the artifact based OMI is resumed in Section 8.3. In the Section 8.3 the dynamic behavior of the OMI is described, since the functionalities that organizational artifacts provide. In this view, the interaction for agents exploiting the OMI is discussed, relatively to the case of artifact use and observation. Finally, Section 8.4 concludes the chapter by discussing strengths and drawbacks of the proposed approach.

8.2 Organizational Artifacts

This section describes Organizational Artifacts (OAs) as those special artifacts used to build the OMI inside the agent work environment. From a developer point of view, OAs are those abstraction tools adopted to conceive the OMI at design time and to reify it at runtime. From an agent point of view, OAs embed those organizational functionalities to use (and observe) in order to participate in organization activities and to access organization resources possibly exploiting, creating and modifying OMI artifacts and related functionalities at runtime.

OAs are thus deployed inside the MAS in order to instrument its work environment with an artifact based OMI. This view promotes a decentralised and distributed model for the OMI, that is implemented using different artifacts each with specialized functionalities. Differently from other approaches, the organizational services provided by the OMI are not centralized in a unique component but distributed within the workspace. Hence, different artifact types are in charge for the management of a specific aspect of the organization.

Being the infrastructure based on artifact computational model (see Chapter 4 and Chapter 5), each OA provides a set suitable operations and observable properties promoting complex patterns of interaction with agents. Besides, they can be used for enabling, mediating, and ruling over agent interactions through a deontic specification, also tracing and regulating the access to resources, and so on. Most important, each OA provides a concrete mechanism for managing obligations and regimentations inside the system, based on the normative specification handled inside the artifacts. The $\mathcal{M}$oise OML and the derivative norm based language NOPL described in Chapter 7 are chosen here in order to identify a programming model defining the artifact programs.

As proposed in [61], the possibility to define the overall normative specification within a normative language suggested the use of a NOPL interpreter inside every OA. In this view, each envisaged OA type encapsulates a subset of the organization specification as it has been identified by a scope specified within the NOPL program. The global organizational specification defined in $\mathcal{M}$oise is thus decomposed in different NOPL programs, which in turns are fed to an interpreter implemented inside each OAs. The adoption of different NOPL interpreters inside each OA, along with an organizational specification originally placed on multiple dimensions, allow to split the artifacts of the OMI according to the various scopes as defined by the normative program. Each organizational artifact is then initialized with *(i)* the NOPL program automatically generated from the $\mathcal{M}$oise OML and *(ii)* a set of dynamic predicates representing the *institutional facts* related to

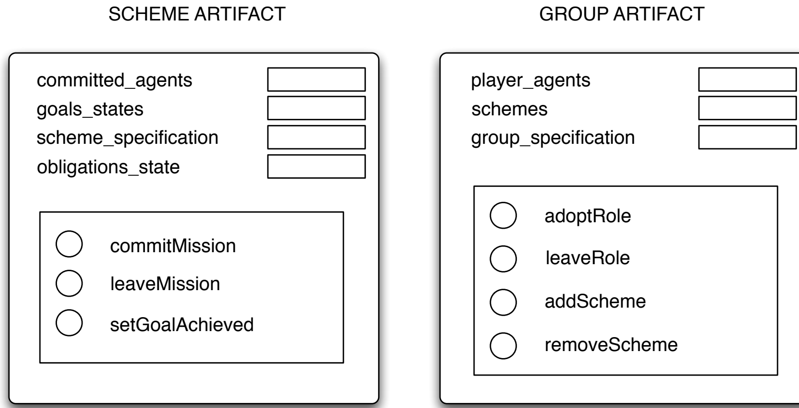

Figure 8.1: Structural diagram for Organizational Artifacts based on $\mathcal{M}$oise OML and NOPL interpreters.

(part of) the organization[2]. The interpreter is then used to compute: *(i)* whether some operation will bring the organization into a inconsistent state (where inconsistency is defined by means of fail states), and *(ii)* the current state of the obligation in their lifecycle.

The current version of the proposed OMI uses two artifact types, related respectively to the functional and structural aspects of the organization. These artifact types are referred to as `scheme` and `group` artifacts, and respectively concern the functional and structural dimensions as expressed in $\mathcal{M}$oise and translated in NOPL. A description of scheme and group artifact is provided in the following sections, while their structural diagrams are showed in Figure 8.1.

8.2.1 Scheme Artifacts

Scheme artifacts are adopted inside the OMI in order to manage instances of the functional schemes and to store the institutional facts related to the their current configuration. Each scheme artifact, or `scheme board`, provides operations by using which a participating agent can notify the organization about its autonomous behavior. In particular, `commitMission` is an operation that can be used by agents to notify their mission commitments, while `setGoalAchieved` can be used to

[2]The term *institutional fact* is here adopted as synonym of organizational fact in order to denote those informational structures representing the actual state of the organization.

acknowledge the organization about the achievement of a a given goal inside the scheme. Link operations are also provided for linking the scheme artifact with other OAs. In particular, `updateRolePlayers` can be linked to update the list agents playing a given role, while `updateMonitoredScheme` can be linked to change the scheme specification to be managed.

A set of observable properties are set in order to show agents some relevant institutional facts related to the functional specification. In particular, a scheme artifact provides the following observable properties:

- `committed_agents` is updated by `commitMission` operation, in order to show the current list of agent commitments;
- `groups` is updated by `updateRolePlayers`, to show which group is responsible for a given scheme;
- `goal_states` is updated inside the `setGoalAchieved` operation, and shows the actual goal states;
- `scheme_specification` showing the functional specification currently managed by the artifact.

By observing or focusing these properties, agents can read the current state of the organizational schemes i.e., in terms of agents committed to missions (committed agents), goal and obligation states, scheme specification etc. Observable properties are automatically updated by the artifact itself, once the operations triggered by the agent complete with success. Operation completion inside OAs is ruled by norms and governed by the NOPL engine through a mechanism which will be described in the next section.

In the context of the hospital scenario, three instances of scheme artifacts are created based on the scopes specified in the NOPL. They refer to the *visitorSch, monitorSch* and *docSch*, as formerly determined by the $\mathcal{M}$oise functional specification. The related OA instances are identified by `visitorSchBoard`, `monitorSchBoard` and `docSchBoard`.

8.2.2 Group Artifacts

Group artifacts are adopted to manage roles and related goals inside a group, according to the specification provided within the $\mathcal{M}$oise structural dimension. At the same time, they are adopted in order to store the institutional facts related to

the current state of the group. Each group artifact, or `group board`, provides operations like `adoptRole` and `leaveRole` by using which an agent can ask the organization to adopt (or leave) a given role inside the group. In addition, it provides the operations `addScheme / removeScheme`, that can be used to add and remove goals for the group, namely to notify the group that is responsible for a new scheme.

The group artifact has a set of observable properties to show current facts related to the actual institutional state of the group. By observing them, agents can read the ongoing structural aspects of the group:

- `palyer_agents` is updated by `adoptRole` operation, in order to show the list of roles actually played by agents inside the group;
- `schemes` showing the schemes for which the group is currently responsible for;
- `group_specification` showing the specification currently handled by the artifact.

Also in this case, observable properties are automatically updated by the artifact itself, once some changes are elicited by operation execution.

In the context of the hospital scenario, two kind of group artifacts are created since the NOPL specification translated by $\mathcal{M}$oise. They refer to the visit group and the staff group, as formerly determined by the structural specification. The related OA instances are identified by `visitorGroupBoard` and `staffGroupBoard`.

8.3 OMI Execution model

Being the OMI an artifact based infrastructure, its dynamics can be described since the transitions affecting the OAs by which it has been built. In what follows, some specific aspects of the functioning of the involved OAs are described. These aspects are detailed on the basis of the activities that agents can perform on OAs, namely: use and perception. The description of the programming model for an agent at work in a workspace instrumented with the described artifact based OMI is provided in Appendix A.

Algorithm 1 Artifact Integration with NOPL (modified from [61])

1: $oe \leftarrow$ current state of the organization managed by the *OA*
2: $p \leftarrow$ current NOPL program
3: $npi \leftarrow$ NOPL interpreter
4: **if** operation op is triggered by agent ag_{id} **then**
5: $oe_{copy} \leftarrow oe$ /*creates a backup of current oe */
6: executes operation op to change oe
7: $f \leftarrow$ a list of NOPL predicates representing oe
8: $r \leftarrow npi(p, f)$ /* runs the interpreter for the new state */
9: **if** $r = fail$ **then**
10: $oe \leftarrow oe_{copy}$ /* the backup state is restored */
11: **signal** $fail$
12: **return** $op_{cs} = \langle \mathsf{op_failed}, op_{res} \rangle$
13: **else**
14: **for all** updated obligations **do**
15: **signal** obligation transitions in r
16: **update** observable properties
17: **end for**
18: **return** $op_{cs} = \langle \mathsf{op_completed}, op_{res} \rangle$
19: **end if**
20: **end if**

8.3.1 Agents using OMI

Inside OAs, the execution of operations is managed in order to respect organizational specifications and maintain global organizational coherence. To this end, an embedded mechanism is adopted inside the OAs to prevent the transition to states that are inconsistent with respect to the actual normative specifications. As seen in Subsection 2.3.3, the basic strategy to prevent undesirable states is exploited by adopting regimentation, by which agents are not enabled to elicit changes in the organizational configuration once these changes violate some regimented norm. Since artifacts operations are the unique means that agents have to change the organizational assets, and thus institutional states, a mechanism is used to wrap operation execution. The mechanism ensures that an operation triggered on OA can be finalized only if the operation outcomes are not eliciting failures according to the NOPL.

To realize the control mechanism, the algorithm 1 is currently processed be-

sides the OA operation execution. It controls whether operation execution respect NOPL organizational constraints, i.e. with respect to ongoing regimentations and obligations. The algorithm is executed inside a single operation step for the involved OA. Operation atomicity fastens that operations that would elicit inconsistent transitions for the OMI does not affect the overall organizational state. Whenever an operation is triggered by an agent, the algorithm firstly stores a consistent copy of the current institutional facts (oe_{copy}, line 5). Once the NOPL interpreter gives a *fail* state the backup state is restored (line 10), while the artifact operation leads to a op_failed which is signalled by the artifact (line 11). This means that the operation triggered by some agent participating the organization is not currently finalized due to some active NOPL norm. In this case the operation completion state is set to op_failed (line 12). This may happen, for instance, whether an action provided by the OA as adopt a role is not permitted due to maximum cardinality for the requested role has been reached. On the other hands, a valid operation is an operation that changes the state of the OMI to one where no *fail* states are entailed by the NOPL interpreter. In case the operation outcome is valid (line 13-19) the algorithm *(i)* signals the obligations transitions collected by running the engine (line 15), *(ii)* updates the current state of the obligations (line 16), and *(iii)* sets the operation completion state to op_completed (line 18).

8.3.2 Agents perceiving OMI

The described process allows to dispatch a series of relevant events inside the workspace—during the application of the algorithm 1. Besides indicating the transitions of the organizational entities, these events have the additional feature to be possibly perceived by agents which are actually observing the related artifacts. In fact, agents engaged in focusing activities and registered to the observability map are notified by all the events occurring on the scrutinized artifacts (agent perception is formalized in terms of operational semantics in Section 5.3). In so doing agents succeed to be acknowledged by the infrastructure, whether, for instance, a norm is being violated. This accounts the property for the OMI to be simultaneously used and perceived by agents. The effect is the possibility for agents to be asynchronously notified by the infrastructure and accordingly react—in order to handle the events of interest. Perceiving events signalled by the OMI is in charge, for instance, of organizational agents, which may adopt the required countermeasures when a norm has been violated. It could be the case of staff agents in the hospital scenario, which are assumed to trigger their sanctioning and rewarding activities since the unfulfillment of norms *n4* and *n5* specified in the

Moise deontic dimension (see Table 7.2 in Subsection 7.2.3). Notice in particular that the Moise norms *n4* and *n5* are first translated into NOPL obligations for goals, as the ones showed in Table 7.5. Norms for goals are then handled inside the OMI by the described algorithm, which in turns allows the OMI to periodically check whether the compliance to that norms is respected or not according to the facts stored in the NOPL.

8.4 Final remarks on Organizational Infrastructures

The programming model introduced in this chapter allows to implement Organizational Management Infrastructures (OMI) as an artifact based facility inside the MAS work environment. The approach is based on existing work on artifact based organizations recently proposed by Hübner et al. [63, 61, 62]. With respect to the work already presented, in this chapter the model has been modified taking into account the computational model of CArtAgO described in the first part of the thesis. In particular, organizational artifacts have been rooted to the semantic of operation execution introduced in Chapter 4 and Chapter 5, including a revised model based on events detailing agent - artifact interactions. By explicitly managing norms and institutional states, this pattern of integration promotes the use of a normative programming language (NOPL) to rule over agents' behavior. In particular, the approach realizes the regimentation of agent actions by regimenting, in this case, operations on decentralised organizational artifacts. Each artifact implementing the OMI is indeed in charge to handle a particular aspect of the organization, where each different scope can be related to a particular dimension as they have been specified, at an higher level of abstraction, by a OML model (i.e. Moise).

It has to be considered that, as far as it has been conceived in Chapter 7, the organizational specification does not cover the overall aspects needed to regulate a complex organization. The original organizational specification is – intentionally – placed in abstract terms, in order to ease the task for the organization designer. This allows to range over many different situations at an application level, and requires little maintenance over time. But, as a consequence, the abstraction level kept at design time is also maintained at application level, thus resulting in few shortcomings in the services provided by the OMI.

A first missing dimension is the one addressed at regulating the dialogic aspect of agent interactions. This aspect is the pivotal one in Electronic Institutions [48] (resumed in Subsection 2.2.4), where, in order to normatively regulate agents

interactions and establish commitments in open environments, the modeling approach envisages a dialogic framework characterized by communication protocols (illocutions) and domain concepts (ontologies).

An important aspect missing in the specification provided in $\mathcal{M}$oise is that it says nothing about "how" the agents should fulfill their goals in practice. This originates a series of important remarks. Consider for instance the mission *mPay*, related to the norms *n*4 and *n*5. *n*5 refers to an obligation for an agent playing the role patient to commit to the mission *mPay* within 5 minutes. Besides, *n*4 allows agents playing the role escort to commit to the same mission. The functional specification in the case of the mission *mPay* only relates to the goal "pay visit". Nothing is specified inside the organization about the choice for patients to delegate the fulfillment of this goal to their escorts. The decision about how to divide the work inside the visitor group is thus left to agents' autonomy and, in this case, is out of the scope of the organization.

Another missing aspects is about the declarative aspects of goals. In $\mathcal{M}$oise a goal is expressed by a simple label, thus the organization is rather vague in describing (i) *how* the obligation for the patient should be fulfilled in practice. (ii) *which* brute state has to be reached in order to consider the goal as achieved. In turn, the $\mathcal{M}$oise model is *not* concerned with an explicit and declarative representation of goals, (i.e. "goals to be" [136]) which declaratively describes a state of affairs that has to be realized in order to consider the goal as achieved. Similarly, the normative model treated by NOPL inside the OMI is not fully capable to specify norms of the type "ought to be" [30, 38, 135], namely norms that explicitly refer the state of the world that has to be realized in order to consider the norm as fulfilled. Again, the organization here demands these aspects to the agents, which are in charge to decide either which are the pragmatic actions to execute in order to achieve a given goal g_i, either whether the goal g_i has been achieved or not. According to the model, the OMI has no way to automatically infer the fulfillment of the goal since the realization of a given state in the working environment. An achieved goal is in fact *notified* to the organization, through an operation provided by the OMI which semantic is "set the goal g_i as achieved". A similar mechanism is envisaged for agents to update their mission commitments, which are notified to the organization by means of operation like "set the mission m_i as committed".

In order to reconcile the organization with the environment where agents work, namely in order to compound institutional and brute states inside the system, the environment aspect has to be introduced as an integral part of the infrastructure. The environment dimension has in charge the aspects related to the interaction space, thus providing, form the one side, concrete infrastructure for agents to

fulfill their objectives, and, on the other side a concrete workplace (referred as brute space) controllable by the organization in order to further apply institutional measures as regimentations and obligations.

Taking aside the dialogical aspect, in the next part of this work a proposal for integrating the environment dimension within the proposed OMI is provided. It assumes to introduce an explicit management of environment as an integral part of the global system, to be put aside the organizational entity as a complementary layer. To not lose the abstraction at which the organizational specification is provided, the new environment dimension is left as independent as possible with respect to the one provided within the organizational specification.

Part IV

Agents, Organizations, Environment: a Unifying Approach

Chapter 9

Embodying Organizations in MAS Work Environments

The adoption of an artifact based OMI constrains agents to be aware of complex structures and constructs proper of organizational specifications: agents must know and manipulate low level primitives related to groups, roles, missions and norms which may be not proper of an application domain. Besides, as far as organizational infrastructures have been modeled, a weak support is given for monitoring and controlling at the organizational level other environment resources deployed inside the same work environment. To bridge this gap, in this chapter the notion of Embodied Organization is introduced as a programming model aimed at reconciling organizational and environmental dimensions. It includes a programmable layer for specifying functional links between organizational entities and environmental ones. In doing so, the possibility to conceive environment infrastructures aimed at better situating the organizational entities inside work environments will be devised. The envisaged computational model marks a clear separation of concerns between the OMI and the other environmental infrastructures. At the same time, the model enables functional relations between the two systems and promotes multiple interaction styles between organizations, agents and their environment.

9.1 Situating Agents and Organizations in Artifact Based Work Environments

A suggestion for achieving a coherent model for organizations of agents situated inside a work environment comes from the way in which humans exploit organized spaces. Typically, a human visitor entering in an organized environment has no need to directly communicate with the organization which is operating behind the scenes, nor he/she needs to explicitly know how such organization is structured. Indeed complex environments where human beings live are often conceived as complex ecosystems of services, being instrumented with media, resources, and concrete objects aimed at assisting and easing individuals in their purposive activities. This approach allows human users to be "unaware" of participating to complex organizational patterns. Besides, it allows individuals to abstract away from the complex patterns required to fulfill works once entering in unknown environments[1]. This is why, for instance, visitors entering in an hospital don't need to plan from scratch complex means-end strategies to fulfill their purposes, or personnel staff doesn't need to explicitly notify their work to the organization. Thanks to the infrastructure, there is no need to directly communicate with the hospital as an organizational entity, nor the need of a complete knowledge of the multiple organizational constructs that are in action behind it. Indeed, hospitals, as any other human organized space, typically deploy several facilities in order to promote the fulfillment of participant expected behaviors.

Following this idea, the contribution of this chapter aims at bridging the gap between actual agents, environments and organizational models in MAS, and at providing an unifying approach addressed to the programming level. Being in an instrumented work environments means – for computational agents as well as for human visitors in the hospital – to not be forced to explicitly interact with the organization as a complex entity, nor to have an explicit representation of the organization in mind in order to exploit its services. To this end, the described model considers environmental and organizational entities as different aspects of organizational work design. Once agents, environments and organizational entities are assumed as different aspects, the designers (and the agents) face a twofold interaction medium. On the one side there is the need to face with the organizational entity, where agents adopt roles, commit missions, coordinate themselves for fulfilling joint activities and workflows; on the other side, there is the need to model a

[1] A similar concept of "intelligent use of space" has been proposed in the context of Artificial Intelligence by David Kirsh [76].

physical, *embodied* environment[2], where agents perform actions, perceive events, communicate, move, access shared resources etc. Although their concerns are similar, their reification usually leads to different approaches: dedicated (middleware) layer in the case of organizations, heterogeneous entities exploitable by agents in the case of environments.

The proposed approach assumes agents' worlds to be instrumented by concrete Environment Management Infrastructures supporting agents with a set of environment resources and functionalities, and by Organizational Management Infrastructures, providing organizational resources and functionalities. This approach leads to the notion of Embodied Organization, which major concern is to conceive a programming model to embody organizational entities into environment ones. In particular we here provide a programming model according to which heterogeneous agents can concurrently operate in the same work environment being supported, coordinated and controlled by a unified infrastructure, aimed at promoting both individual and collective activities.

The chapter is organized as follows. In Section 9.2 the notion of Environment Management Infrastructure is introduced as an environmental extension of the organizational infrastructure. It devises a structured approach to environment as modeled in terms of decentralised environmental artifacts aimed at transparently mediating organizational functionalities. Section 9.3 proposes a programming model by which functional relationships can be established between environment and organizational entities. The model makes use of the notion of constitutive rules and implements it thorugh event based workspace rules discussed in Chapter 6. Section 9.4 concludes the chapter by discussing related works and remarks.

9.2 Environment Management Infrastructures

As far as the OMI model has been conceived, in order to exploit the organization agents need to know how to exploit organizational artifacts—which are the basic building blocks of the OMI and provides its interfaces. Besides, agents require not obvious capabilities to bring about organizational and institutional notions (i.e. role, missions, schemes, norms, obligations etc.) which typically are not native constructs of their architectures. This result in a strict requirement for agents programming models, which in turns must be aware of organizational artifacts (OA) and their complex functioning in order to exploit them. Even more, in order

[2] *Embodied* has to be intended as *reified entity of agents computational world*, more than in its cybernetic notion of having a physical body.

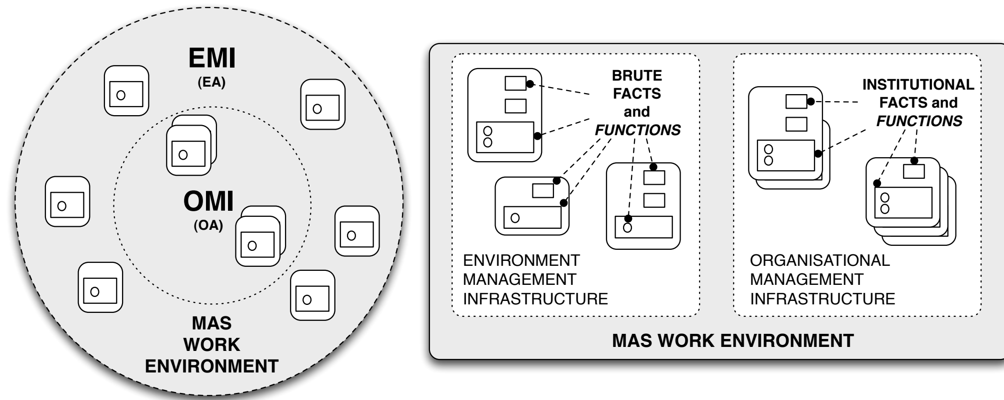

Figure 9.1: Two different perspectives of Environment and Organizational Management Infrastructures in MAS. (Left) They are logically separated systems, integrated in the same work environment. (Right) The EMI provides either brute facts and functions, the OMI provides either institutional facts and functions.

to exploit the organizational infrastructure, agents need to perform a set of additional activities which are not concerned with the fulfilment of their purposes. On the contrary, as far as the OMI is designed, these additional activities are needed to explicitly notify the organization about changes affecting the whole environment. For instance, an agent who wants to adopt a role needs to explicitly use an `adoptRole` operation upon a particular OA, as well as an agent who achieves a goal needs to explicitly notify it with a `setGoalAchieved` operation.

In order to bridge this gap, the proposed model explicitly introduces an environment support inside the system. We refer to Environment Management Infrastructure (EMI) to indicate the whole set of environmental entities aimed at supporting agent activities besides the organizational entities. In this view, Environment Management Infrastructure and Organization Management Infrastructures are situated side by side inside the same MAS work environment, but, most important, they are conceived as separated systems. They are assumed to face distinct problem domains, the former being related to concrete environment services and tools and the latter dealing specifically to organizational services.

Figure 9.1 shows an abstract representation for the scopes supplied by the two systems. In particular, the figure resumes an abstract architecture for a work environment once it is conceived in terms of both environmental and organizational infrastructures (Figure 9.1, left). As emphasized many times along this work, arti-

facts embed a twofold function in MAS. They can be used for epistemic objectives (storing observable states) and for pragmatic objectives (providing a usage interface to be used to trigger the execution of operations). This twofold functionality is then exploited either by agents, which are assumed as exploiters, either by the artifact developer, which may design artifacts in order to provide epistemic and pragmatic services as well.

In conceptual terms, the twofold functionality supplied by one single artifact can be recasted in terms of artifact infrastructure, once a composition of several artifacts is set in order to provide a facility inside the work environment. Even in the case of an artifact based infrastructure, the two functionalities remain independent each other. As a single artifact, an infrastructure provides observable states, namely machine readable data aimed at being perceived by agents for improving their knowledge base. Besides, as a single artifact, an infrastructure provides operations, namely process oriented services aimed at being exploited by agents for externalising activities in terms of external actions. Thereby, the epistemic functionality of artifacts can be shifted to the informational dimension to which the infrastructure is addressed. This relation between functions and brute phenomena is also emphasized in the taxonomy of facts discussed by Searle ([128], p. 121)—where functions has to be intended in their general, non mathematical, notion of functionalities. Once addressed to the brute reality, functions are "performed solely in virtue of causal and other brute features of the phenomena". Artifacts implementing functions are thus providing means for agentive activities (*"this is a screwdriver" or "this is an automatic desk"*). Searle emphasizes also the relation between institutional facts and their (status) functions, which can be intended as functions performed (by agents, typically) only by way of social agreement and collective acceptance (*"this is a twenty dollar bill" or "this is an applicable role"*).

Taking the perspective of Searle in the context of MAS work environments, the EMI can be deemed as a container of *brute facts*, namely states which are related to the concrete, "physical" workplace where agents live (Figure 9.1, right), while the OMI can be assumed as a container of *institutional facts*, namely facts related to the organizational states. On the other hand, the pragmatic functionalities of artifacts can be related to the operational dimension to which the infrastructure is addressed. In so doing, the EMI can be deemed as a provider of *brute functions*, namely operations which can be used by agents to exploit causal functionalities, i.e. in order to change the concrete workplace or fulfill goals (i.e., *"using a screwdriver" or "using an automatic desk"*). Being the OMI an infrastructure explicitly conceived with the aim to provide organizational functionalities which

agents exploit thanks to collective acceptances, it can be assumed as a provider of *institutional functions*, namely operations which can be used to change / update the institutional facts inside the organization (*"paying twenty dollars" or "adopting a role"*).

Before providing a unifying approach and explaining how the two infrastructures can be functionally related, the following sections describe how an artifact based EMI can be implemented in practice making use of the artifacts and their computational model.

9.2.1 Shaping Environment Management Infrastructures on Organizational Entities

A specific infrastructure is assumed to realize inside the work environment an environmental support for the organizational entity. Environment Management Infrastructure (EMI) is aimed at bridging the gap between agents and organization, in particular by mediating the interactions between agents layer and organizational one. Whether the objective is to provide an environmental support to the organization, the EMI can be shaped based on the dictates of an organizational specification.

Using the same approach adopted in Chapter 8, we will use the artifact as main abstraction tool to implement the EMI. In particular, artifacts are adopted to provide a concrete (*brute*) ground – at the environment level – to the organizational infrastructure. Symmetrically to the case of the OMI, we refer to the artifacts used to build an EMI as Environmental Artifacts (EA). Besides artifacts, an EMI can further make use of workspaces, i.e., in order to model a notion of locality in terms of an application domain.

9.2.2 Environmental Artifacts

Since a $\mathcal{M}$oise functional specification it is quite straightforward to find a basic set of artifacts and workspaces fitting the making of an EMI. Figure 9.2 shows how they are identified in the case of the hospital scenario (introduced in Chapter 7). In particular, taking an agent perspective, the developer here simply imagines which kind of service may be required for the fulfillment of the various goals. In doing so, designing an EMI is not dissimilar to instrument a real world in the human case: (*i*) to model the room it will be used a specialized *workspace*, (*ii*) to automate bookings it will be provided a *electronic desk* artifact, (*iii*) to finalize

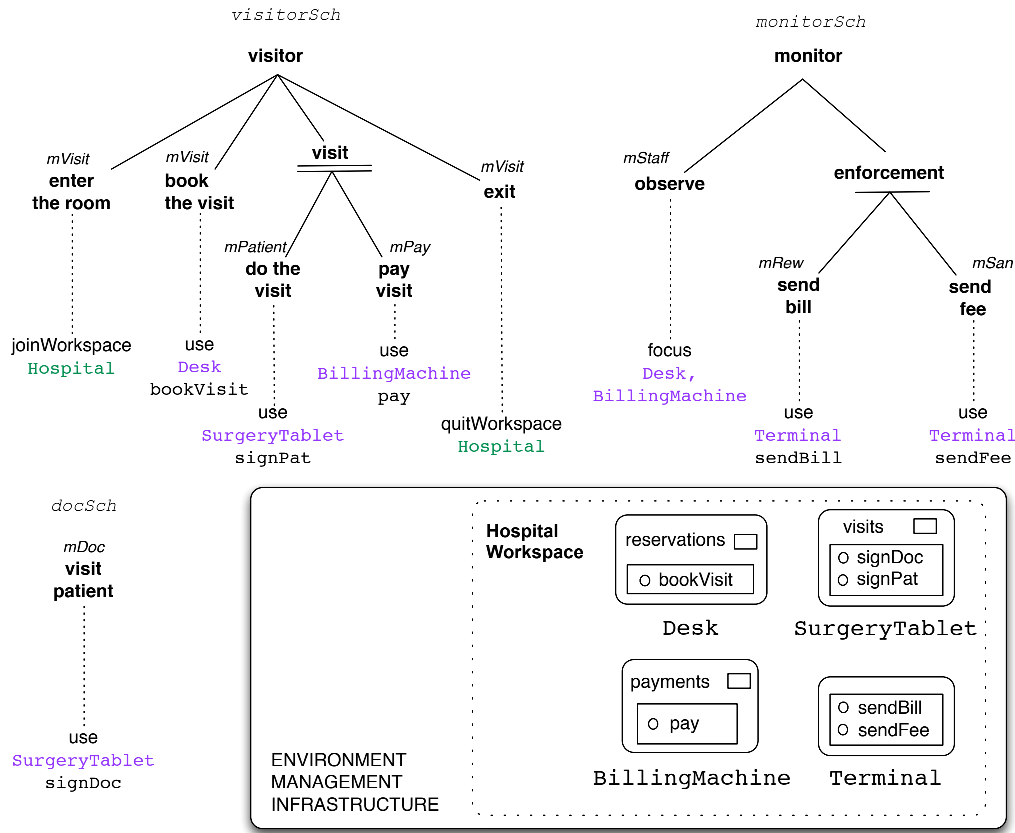

Figure 9.2: Artifacts building an Environment Management Infrastructure are identified since a Moise functional specification (cfr. Subsection 7.2.2). Out of the box, artifacts and their operations are mapped into the leaf goals of the functional schemes (Time-to-fulfill values are omitted for simplicity). In the box, all the entities involved in the EMI are showed through their structural representation.

visits it will be provided a (program running on an) *electronic tablet* artifact, (*iv*) to automate payments it will be provided a *billing machine* artifact, and (*v*) to send fees and bills it will be provided a *terminal* artifact.

Besides the abstract indication of the different artifacts exploitable at the environment level, the Figure 9.2 also shows the actions to be performed by agents for the fulfillment of the various goals. Those actions are directly mapped into artifact operations (i.e., functions), or addressed to the involved workspace: for

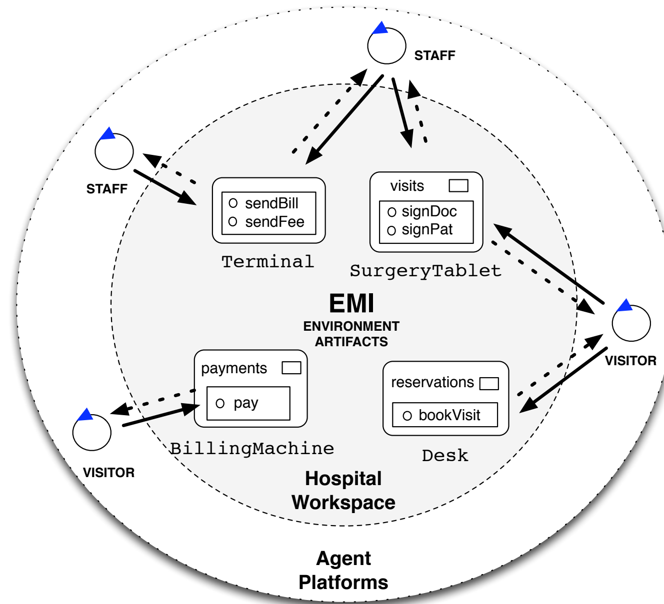

Figure 9.3: Agents at work inside the work environment instrumented, in terms of artifacts and workspaces, with an Environment Management Infrastructure shaped on the organizational specification of the hospital scenario.

instance, to enter the room (which is related to the *mVisit* mission) the action `joinWorkspace("Hospital")` has to be performed; to book the visit (again it is related to the *mVisit* mission) the action `use("Desk", bookvisit)` has to be performed on the desk artifact; to monitor and observe the patient activities (related to the *mStaff* mission) a couple of `focus` activities can be initiated, respectively on the desk and on the billing artifacts, etc. Notice, in this latter case, that focusing EA is an alternative strategy exploitable by staff agents to recognize violations of norms—beside the direct focusing of the OA signals which has has been described in Subsection 8.3.1. Instead of perceiving OMI events signalling violations of goal norms, staff agents can in this case directly focus the environment infrastructures which have been deployed for fulfilling goals.

Figure 9.3 shows the interaction model between agents and artifacts inside the EMI. Notice that some of the envisaged artifacts can be used for different purposes by different positions (i.e., roles) inside the organization. For instance the electronic tablet can be used either by the doctor either by the patient to finalize

a visit, as well as the terminal can be used by the staff agent either for sending fees, either for sending bills, and so on.

In what follows the entities involved in the EMI are better detailed in terms of their computational model.

Hospital Workspace

An Hospital workspace is set as a logic container for both agents and infrastructures, thus providing a notion of locality to be adopted at an application level. In computational terms, the workspace indeed resembles the notion of "room", as it has defined in the hospital scenario. Entering and leaving the room in this case is assumed, in terms of software agents, as joining and quitting the hospital workspace.

Electronic Desk

A `Desk` artifact is modeled as an application providing the services of an electronic desk, namely an automatic machinery allowing users to achieve their book the visit goals. For this purpose an operation is available in the artifact interface `bookVisit(Params)`, completing in a single step and returning as operation feedback the reference code for the visit. The `Desk` also provides an observable property `reservation(Value)`, containing the set of reference codes already booked[3].

Surgery Tablet

A `Tablet` artifact is modeled as an application running on a smart laptop providing users with the possibility to finalize medical visits. The tablet is assumed in this case as the tool supporting the fulfillment of both the goals related to a visit, by assuming each visit being finalized by a couple of signatures to be provided either by the doctor and by the patient[4]. To this end, two single step operations are provided, `signDoc(Params)` being used by the doctor and `signPat(Params)` being used by the patient. The `Tablet` also provides an observable property

[3]Parameters to be fed to artifact operations and values expressed by their observable properties are here omitted for simplicity.

[4]In this case, do the visit is the goal to be fulfilled by patients, and visit patient the goal to be fulfilled by doctors. Of course, the activities allowing the fulfillment of a medical visit may be clearly more complex, in a real scenario, than the ones described here. A series of assumptions and abuses are made just to exemplify the use case.

`visits(Value)`, containing the set of reference codes for the visits already fulfilled.

Billing Machine

A `BillingMachine` artifact is modeled as an application providing payment services, namely an automatic machinery allowing users to fulfill their pay visit goals. For this purpose an operation is available in the artifact interface `pay(Params)`, completing in a single step and returning as operation feedback a payment receipt. The `BillingMachine` also provides an observable property `payments(Value)`, containing the set of reference codes for the visits already paid.

Terminal

A `Terminal` artifact is modeled as an application providing the functionalities needed for achieving the goals of a staff agent, namely sending fees and bills to patients. The `Terminal`, in this case, provides no observable properties but two operations are in its usage interface, namely `sendBill(Params)` and `sendFee(Params)`. They are implemented as single step operations, being used by the staff agent respectively to achieve the goals related to rewarding and sanctioning missions.

Notice that all the artifacts involved in the described EMI are self-contained, thus without the need to be linked each other. Different application domains may require to link together the artifacts, or to control them by the mean of workspace rules, as described in Subsection 4.3.1 and Section 6.5. On the other hands, each of the above mentioned artifacts could be enriched with a manual document, containing the operative instructions to be followed by agents in order to exploit them. As explained in Subsection 4.5.2, manuals allow agents that do not know artifact specifications at design time, to integrate usage protocols in order to learn how to use their functionalities at runtime. Besides providing a suitable mechanism for easing matchmaking, this turns to be a pivotal aspect for improving the openness of the whole system.

9.3 Relating Organizations and Environments

As far as the approach has been presented in the previous section, the MAS work environment is conceived as a workspace instrumented with a twofold decentralized infrastructure: an OMI, specialized in organizational functionalities and in

controlling agents behavior, and an EMI allowing agent activities and situating the organizational control inside the work environment. By ruling agents behavior OMI achieves coordination, promotes cooperation and prevent deviation from equilibrium and undesirable states. By providing resources to be exploited by agents in a seamless fashion, the EMI is conceived as a reliable set of artifacts allowing agents in achieving their goals.

It is important to remark that the environment infrastructure is not supposed to "hide" the organizational one. Agents having the required capabilities to directly exploit OMI services (i.e., organizational agents in the description given in Subsection 2.3.3, staff agents in the hospital scenario) are not prevented to directly use organizational artifacts, thus having a direct access their functionalities (see Figure 9.4). On the other hands, the EMI is explicitly conceived so as to mediate between the agents participating the organization and the organization itself. The mediation is tackled by establishing functional (programmable) relationships between the environmental entities and organizational ones, as the next sections detail.

9.3.1 Establishing functional relations between organizations and environments

For embodying organizations into environments the definition of concrete relationships between environmental and organizational infrastructures has to be provided. For doing this, the abstract principle at the basis of *constitutive rules* is adopted. Introduced in social sciences by Searle [128], constitutive rules have been originally adapted in a formal framework for (normative) MAS by Boella and van der Torre [9]. According to Searle theory, constitutive rules are of concern when:

> Collective intentionality assigns a new status to some phenomenon, where that status has an accompanying function that cannot be performed solely in virtue of the intrinsic physical features of the phenomenon in question. This assignment creates a new fact, an institutional fact, a new fact created by human agreement ([128], p.46).

In Searle's theory, this special assignment is a constitutive rule of the type:

$$X \quad count-as \quad Y \quad in \quad C$$

where the relation "count-as" is the operator for the assignment of the new status function Y to X, when the context is C. In other terms, the application of a

constitutive rule introduces, with Y, a new status for the facts in X. This new status gives to X some features that it does not already have by satisfying solely its physical property. This new status, then is reachable only if the context is C, and, most important, if there is a collective agreement or at least some acceptance for the value assigned to Y. For instance, a piece of paper may count as money once responding to certain physical characteristics and thanks to a certified authority; saying yes in front of a priest during a religious ceremony may count as being married, and so on.

"Count-as" can also be viewed as a special relation between the effects of an action performed by an individual in a non institutional context and those additional effects – produced by that action – which can be addressed to the institutional context. An action performed by an agent in its environment constitutes, or "count-as", a conventional or an institutional action once it is situated in a particular institutional context. A given action performed by an agent acquires an additional *effect* (e.g., "count-as" empowerment), due to the fact that the institution may recognize the institutional meaning for an ongoing activity and thus ascribe a normative outcome to it.

Adapting constitutive rules in MAS, in the context of our instrumented work environment, allows to define a bidirectional relation. The "count-as" effect can be assumed, in this case, as a *vehicle* to route both relevant facts (and events) to institutional ones (see Figure 9.5). Thus, using – or attempting to use – an EA operation may produce events being, from the point of view of the organizational entity, a "count-as" for an additional institutional event[5] . For instance, in the hospital scenario, finalizing a pay operation on a billing machine may count-as having achieved the goal pay visit; focusing a billing machine may count as adopting the role staff; leaving the room without fulfilling the payment may count as violating a norm, and so on.

Besides count-as, a second relation can be considered, that is an enactment effect dealing with regimentation (or enforcement) aspects that the organization may provide in order to control the system. Symmetrically to count as, the "enact" effect can be assumed, in this case, as a *vehicle* to route institutional facts (and events) to environmental ones, thus establishing a control loop from the organization to the environment dimension of the system. Using enact relation, the organization thus is aimed at producing a normative control by enacting changes

[5] I would to explicitly thank Antônio Carlos da Rocha Costa and Jomi Fred Hübner for having discussed with me these aspects at Ecole Nationale Superieure des Mines in Et-Etienne, in February 2009.

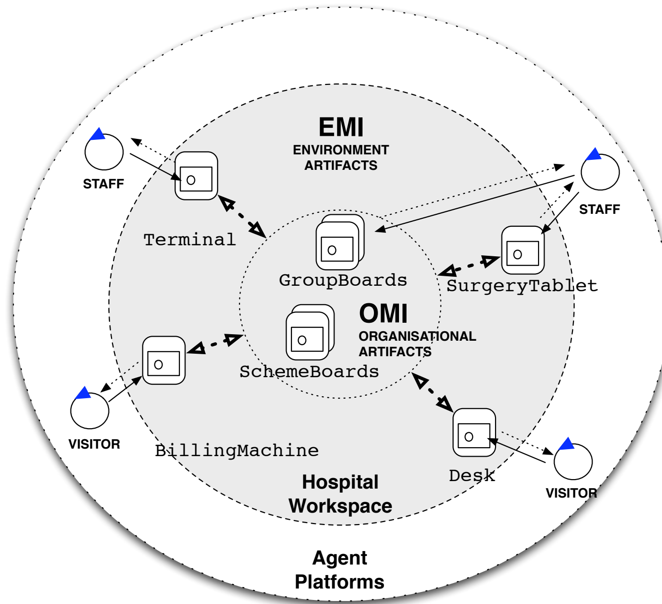

Figure 9.4: Agents at work in environments instrumented with EMI and OMI. Using artifact operations and perceiving their observable properties agents have access to both institutional and brute dimensions of the system.

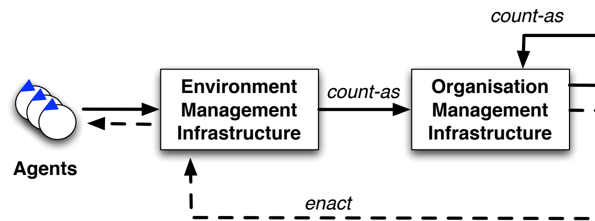

Figure 9.5: As functional programmable relation between EMI and OMI, constitutive rules allows environmental entities to mediate between agents and organization.

upon the environment, in order to promote desirable states of equilibrium (see Figure 9.5). In the hospital scenario, for instance, an institutional event signalling that a group is not well formed may enact the regimentation of a related environment functionality. For instance, an event signalling the reached maximum cardinality for the visitor groups (N_{VMAX}) may enact disabling the book visit operation on the desk. Violating the obligation imposed to the staff agent to fulfill sanctioning and

rewarding missions may enact the expulsion of that agent from the workspace.

9.3.2 Embodied Organization Rules

By assuming the relations between organizational and environmental entities as governed by constitutive rules, it is now possible to specify a programmable relations between EMI and OMI:

Definition 1 *(Embodied Organization Rules)* The programmable relations between organizational and environmental infrastructures are defined in terms of *Embodied Organization Rules* as specified in Table 9.1.

Structures defining *Emb-Org-Rule* refer to *count-as* and *enact* relations.

Count-as rules, state which are the consequences, at the organizational level, for a specified event generated inside the overall infrastructure. In this case, *count-as* rules indicate how, since the actions performed by the agents, the system automatically detects relevant events, thus transforming them to some additional event to be processed at the organizational level. In so doing, either relevant events occurring inside the EMI (possibly triggered by agents actions), either events occurring in the context of the organization itself (OMI) can be vehicled to the institutional dimension (see Figure 9.5): these events can be further translated in the opportune institutional changes inside the OMI, that is assumed to update accordingly.

Enact rules, state, for each institutional event, which is the control feedback at the environmental level. Hence, *enact* rules express how, since the specified *Emb-Org-Rule*, the organizational entities are assumed to control the environmental ones. The use of enact rules allows to recast organizational events (i.e. role adoption, mission commitment) in order to elicit changes in the work environment.

The pivotal aspects to be considered is that both enact and count-as rules are grounded to the events generated inside the workspace—which in this particular case consist in events originated inside the whole work environment (*we-ev*). Those events are then specialized once based on organizational events (i.e., *omi-ev*, describing those relevant changes occurring inside the institutional layer in terms of OMI events) and on environmental events (i.e., *omi-ev*, describing those changes occurring inside the environment layer in terms of EMI events).

$Emb-Org-Rule$	$::=$	$\langle count-as \rangle \quad \vert \quad \langle enact \rangle$	Constitutive Rules
$count-as$	$::=$	$\langle we-ev \rangle \longrightarrow \langle omi-ev \rangle$	Count as Rules
$enact$	$::=$	$\langle omi-ev \rangle \longrightarrow \langle emi-ev \rangle$	Enact Rules
$we-ev$	$::=$	$\langle omi-ev \rangle \quad \vert \quad \langle emi-ev \rangle$	Work Environment Event
$emi-ev$	$::=$	$\langle ws-ev \rangle$	EMI Event
$omi-ev$	$::=$	$\langle ws-ev \rangle$	OMI Event

Table 9.1: Definition of Emb-Org-Rules in terms of constitutive rules and events.

By assuming the work environment as coincident with a single workspace, both $emi-ev$ and $omi-ev$ can be rooted to workspace events $ws-ev$ (as defined in Table 9.1). Given this, the events of interest for the applicability of Emb-Org-Rule are the events originated inside the workspace hosting the organizational and the environmental infrastructures.

9.4 Final Remarks on Embodying Organizations in MAS

The contribute of this chapter revise and extends on a couple of recent works presenting the notion of embodied organization in MAS programming [108, 107]. In this view, the environment dimension is explicitly modeled and programmed as a specific infrastructure instrumenting the work environment *aside* the organizational ones. A bidirectional functional relation is then estalished in order to relate the twofold infrastructure in a coherent ensemble, based on the notion of constitutive rules.

As discussed in Chapter 3, the application of constitutive rules is not new in MAS area. Among others, the approach provided by Dastani et al. ([33, 34]) makes use of the same kind of bidirectional relations to define what they call count-as and sanctioning rules. Our approach to enact rules is more general, allowing different kind of feedback control at the level of the EMI beside sanction and reward policies[6]. Typically such rules are well suited for regulating norms for prohibitions, that in NOPL are expressed by regimentation. They conversely can

[6] Indeed, many proposals exist in literature providing constitutive rules as a tool for addressing normative dimension inside MAS. A more detailed comparison on these themes can be found in Section 3.2.

be adopted for permissions, i.e. for enabling a certain functionality. It is worth to remark that the normative control that the organization may want to apply over the environment is still specified by NOPL specifications inside the OMI. For example, a domain norm specifying that a certain door "can be used by medical staff only" can be ensured by user's personal badges or keys procedures that an enactment may control. In this example, the opening procedure to open the door is the *instrument* that implements that norm, while such instrument is controlled by enactments. However, norms for visitors like "pay an additional fee" or "No Smoking" do not need to (or cannot) be forced by any enact rule in the same way enact can force agents to access authorized area only. In this case there is no artifact based instrument capable to ensure the norm, and the role of an organizational agent becomes necessary to verify violations and possibly enforce a desired behavior.

The presented approach makes no assumptions on agent architectures and models. In the context of the hospital scenario agents could be even human beings – actually they "should be" human beings, at least in the case of patients – or computational agents embedding some kind of user human machine interface. This accounts, for the interrelatedness of social and technical aspects, to the notion of complex organizational work design as envisaged in the trend of sociotechnical approaches to organizations (see, among others [44, 103]). In fact, some of the aspects already present in the proposed scenario emphasize the relationships between socio and technical elements, once they may lead to the emergence of effectiveness and well-being of individuals. Promoting the "fit" between people and their work in order to optimize well-being and overall system performance resembles the well-known principle of *ergonomics design* [5]. In general, this accounts for workers' capabilities and limitations in seeking to ensure and ease their tasks, tying to fill the gap between objects to be exploited to achieve goals and the human attitudes adopted to bring about them (see, among others, [87, 88, 104]). Similarly to our approach, in ergonomics design a set of facilities are specifically conceived to instrument environments. Accordingly, indications, environment structures, equipments, information and a series of support mechanisms should be straightforwardly afforded by users, and are explicitly designed to instrument workplaces in order to fit the work of individuals. The overall environment is thus modeled as an "habitat" of supporting infrastructures, aimed to suit each individuals and accordingly to promote specific patterns of cooperation, to which users may effortlessly participate.

Chapter 10

Programming Embodied Organizations

After having detailed an abstract model for Embodied Organizations, this chapter details a concrete description on how the approach to can be engineered in practice. Adopting the scenario previously introduced as guideline, the chapter is enriched by a series of concrete examples aimed at providing a practical methodology to developers in programming an embodied organization. A final section analyzes strength and weakness of the proposed approach, discussing the main features with respect to the challenges devised in the first part of the thesis.

10.1 Embodied Organizations in Practice

After having clarified an abstract model for Embodied Organizations, this chapter aims at detailing the approach by discussing the programming approach concretely. To this end, the mechanism for implementing Embodied Organization Rules is identified in terms of workspace events and workspace rules. Then, the hospital scenario introduced in Chapter 7 is implemented, including both environmental and organizational infrastructures which are then related each other using "count as" and "enact" relationships. The description also analyzes agent models, discussing the different kind of activities required for agents at work in environment instrumented with and without an Embodied Organization. The benefits and drawbacks of the proposed approach are pointed out, also enlightening the main differences with related approaches.

In Section 10.2, an example of programming count as and enact rules is pro-

vided, describing an Embodied Organization applied to the hospital scenario. Section 10.3 takes an agent perspective to show the different programming models for agents operating in the hospital workspace. Two different agents are considered playing the same role, either situated in organizational infrastructure (OMI) and in a Embodied Organization (OMI+EMI). In Section 10.4 a series of relevant remarks and discussed along with the main limitations identified for the proposed approach. Finally, Section 10.5 sum up the contribution with final discussions and remarks.

10.2 Programming Embodied Organization Rules

Placing count as and enact rules in terms of events and assuming the whole work environment as included in a single workspace allows to frame, at a programming level, the the problem of the implementation of Emb-Org-Rule in terms of workspace rules—as they have been described in Chapter 6.
Formally: let $\langle Ag, Ar, Art, Ev, M, R, t \rangle$ be the configuration of the workspace as described in Chapter 5, where R represents the set of workspace rules and both EMI and OMI artifacts are included in the set of artifacts Ar. Being:

```
+trigg_ev :  context -> body
```

the specification of a w_rule, and being $\langle r_{id}, w_rule \rangle$ an associated entry in R, then a Emb-Org-Rule can be expressed in terms of workspace rules.

The equivalence between workspace rules and Emb-Org-Rules follows by their definitions: hence, once a workspace coincides with the work environment hosting the overall infrastructures, Emb-Org-Rules coincide with workspace rules.

In order to perform a proof-of-concept of the system, a series of Emb-Org-Rule are described in terms of workspace rules and applied in the context of the hospital scenario.

10.2.1 Programming Count-as Rules

Let us consider that the organization expects that an agent joining the hospital workspace is identified by va_id and is assumed to play the role visitor, which purpose is to ask for a medical visit and possibly achieve it—as specified in $\mathcal{M}$oise-OML functional specification. Given that assumption, an event join_req, $\langle va_{id}, t \rangle$, dispatched once an agent identified with va_{id} tries to enter the workspace, from

```
+join_req(Ag)
: true
->  make("visitorGroupBoard",
"OMI.GroupBoard",
["moise/hospital.xml","visitGroup"]);
     make("visitorSchBoard",
"OMI.SchemeBoard",
["moise/hospital.xml","visitorSch"]);
     apply("visitorGroupBoard",
adoptRole(Ag, "patient"));
     include(Ag).

+op_completed("visitorGroupBoard", _,
        adoptRole(Ag, "patient"))
: true
-> apply("visitorSchBoard",
commitMission(Ag, "mPat")).
```

```
+ws_leaved(Ag)
: true
-> apply("visitorGroupBoard",
        leaveRole(Ag, "patient")).

+op_completed("BillingMachine",
    Ag, pay)
: true
-> apply("visitorSchBoard",
    setGoalAchieved(Ag, pay_visit)).

+op_completed("Terminal",
    Ag, sendFee)
: true
-> apply("monitorSchBoard",
    setGoalAchieved(Ag, send_fee)).
```

Table 10.1: Example of count as rules in the hospital scenario.

the point of view of the organization "count-as" creating a new position related to the visit group. In other terms, we are interested in making the event join_req to "count as" va_{id} adopting the role visitor. This relation is specified by the first rule in Table 10.1 (left): such a rule states that since an event signalling that an agent *Ag* is joining the workspace, an *Emb-Org-Rule* must be applied as an instance of workspace rule. The body of the rule specifies that two new instances of organizational artifacts related to the visit group must be created using the make operator. In this case the artifacts to be created are identified by `visitorGroupBoard` and a `visitorSchBoard`. The following operator is specified in order to constitute the new role inside the group: apply acts on the `visitorGroupBoard` artifact just created by automatically making the agent *Ag* to adopt the role patient. Finally, once the adopt role operator succeeds, the last operator includes the agent *Ag* in the workspace. A similar count as rule can be specified to create organizational artifacts in the context of the staff group and start agents to play a role accordingly. For instance, entering in a hospital workspace for a staff agent may elicit the creation of the related OAs, as `staffGroupBoard`, `monitorSchBoard` and `docSchBoard`, while the role doctor can be adopted as an additional effect. In the above described scenario, the effect of the application of whole rule provides an institutional outcome to the joinWorkspace actions—which in normal conditions

only allows an agent to enter the workspace. Joining the workspace is in this case replaced by a sequence of Emb-Org-Rule, operators stating what this event means in organizational terms.

When the effects of the role-adoption are committed, as previously described, a new event is generated by the group board: $\langle$op_completed, $\langle$"visitorGroupBoard", va_{id}, adoptRole, patient $\rangle\rangle$. For the organization, such an event may "count-as" committing to mission *mPat* on the visitorSchBoard. This relation is specified in the second rule specified in Table 10.1, where a commitMission is applied to the visitorSchBoard for the mission *mPat*.

Similarly, an event $\langle$ws_leaved, $\langle va_{id}, t\rangle\rangle$ signalling that the visitor agent has left the workspace "count-as" leaving the role patient. This relation is specified in the first rule in Table 10.1 (right), where a leaveRole is applied to the visitorGroupBoard for the role patient. . At the same time, an event like the one described by $\langle$op_completed, $\langle$BillingMachine, va_{id}, pay, $t\rangle\rangle$ signals that a visitor agent has successfully finalized the pay operation upon the billing machine. From the organization perspective such an event "count-as" having achieved the goal pay visit on the visitorSchBoard (second rule in Table 10.1, right). Similarly, an event like the one indicated by $\langle$op_completed, $\langle$Terminal, sa_{id}, sendFee , $t\rangle\rangle$, signalling that a staff agent has successfully used the terminal to send the fee to a given patient, "count-as" having achieved the goal send fee (third rule in Table 10.1, right).

This event based mechanism makes it possible to handle agent activities at a fine granularity, allowing to model agent also in their *attempts* to do some activity. Notice that, wether organizational functionalities are regimented, the execution of an operation upon an organizational artifacts may fail. The completion state of an organizational functionality provided by the OMI is indeed regulated by the NOPL interpreter, which from time to time relies on the actual configuration of the organization in terms on NOPL facts, rules and norms. A failure may happen in the first rule specified in Table 10.1, for instance when the visitor group is not well formed (each visitor group may contain at least one visitor and one patient). In this case, according to the semantic of workspace rules, all the effects of the body are canceled: the two organizational artifacts are removed, the second operator include(Ag) is not applied and the event +join_req(Ag) is wasted. The result is that the join action initiated by *Ag* has no effect on the system, the role is not adopted nor the agent is included the workspace. It is worth remarking, as defined in Table 9.1, that events triggering count as rules can rise either from EAs (as in the case of events occurring in the billing machines and in the terminal) either from OAs (as in the case of role adoption, which is suddenly related to a mission

```
+signal("visitorGroupBoard",
  role_cardinality, visitor)
: true
-> disable("Desk", bookVisit).
```

```
+signal("monitorSchBoard",
  goal_non_compliance,
  obligation(Ag,
   ngoa(monitorSch,mRew,send_bill),
   achieved(monitorSch,send_bill,Ag),
   TTF)
: true
-> exclude(Ag).
```

Table 10.2: Example of enact rules in the hospital scenario.

commitment).

10.2.2 Programming Enact Rules

Besides count as, *enact* effects are defined to indicate how, from the events occurring at the institutional level, some control feedback can be applied to the environmental infrastructure. As far as the execution of the operation has been conceived in Subsection 8.3.1 , the OMI automatically dispatches events signalling ongoing violations. Violations are thus organizational events (*omi-ev*) which suddenly may elicit the application of some *enact* rule. NOPL fail events are indeed a typical application of enact rules that may be envisaged for regimenting the environment once a violation occurs.

shows an example of such mechanism. In this case, a regimentation is installed by the organization thanks to the enact rule stating that an event $\langle \mathsf{signal}, \langle \mathtt{visitorGroupBoard}, \mathtt{role_cardinality}, \emptyset, t \rangle \rangle$ signalled by the `visitorGroupBoard` indicates the violation for the `norm role_cardinality` (such a norm is given in NOPL and described in Section 7.3, Table 7.9). The specification of the enact rule is given in Table 10.2.2 (left), where the reaction to this event is specified in order to enact disabling the book operation on the desk artifact, for all the agents inside the workspace. Notice that the absence of any parameter related to agent identifier in the `disable("Desk", bookVisit)` operator makes the disabling to affect the overall set of agents *Ag*, and the workspace usability map is modified as well.

Similarly, violating the obligation imposed to the staff agent to fulfill sanctioning and rewarding missions elicits the scheme board assigned to the *monitorSch* to signal the event $\langle$signal, $\langle$`monitorSchBoard`, `goal_non_compliance, obligation(Ag, ngoa(monitorSch,mRew,send_bill), achieved( monitorSch, send_ bill,`

Ag), TTF), $t\rangle\rangle$. This event is generated thanks to the NOPL norm goal_non_compliace (explained in Table 7.5, Section 7.3), and, due to the enact rule specified in Table 10.2.2 (right), causes the exclusion for the Ag agent from the hospital workspace.

Notice that the violation of the norm goal_non_compliance is particularly relevant for triggering control activities to be performed by organizational entities. The organizational entity has, in this case, two options to arrange a countermeasure. In fact, besides the application of the enact rules automatically handled at an infrastructural level, goal unfulfillment events could be used to coordinate organizational agents signalling the need to intervene. In this second option, an additional norm may be specified in order to oblige a staff agent to commit to a further recovery mission. i.e. to judge and possibly sanction the ongoing violations.

It is worth to remark that whether workspace rules have been associated to regulatory rules in Chapter 6, in this case the same programming construct resembles the notion of constitutive rules, as it has been defined by Searle in [127, 128]. Thanks to the presence of an organizational entity, that assumes in this context the boundaries of a "social reality", workspace rules define (constitute) an activity the existence of which is logically dependent on the rules themselves. In other terms, thanks the possibility to exploit both a brute and an institutional dimension, as embedded inside the work environment, and due to a (possibly implicit) agreement on their social functions, allows workspace rules do not merely regulate but create and also define new forms of agent behavior. Thereby, the very basic notion of "adopting a role", or "setting a goal achieved", "committing missions" are recasted and redefined in terms of constituive/workspace rules.

An example of agents at work in a workspace instrumented with EMI, OMI and Embodied Organizational Rules is provided in the next section. The example shows in particular the differences between an agent at work in a workspace instrumented with embodied organizational infrastructures only.

10.3 Programming Agents in Embodied Organizations

In this section we describe in practice the different programming models for agents at work in organizations instrumented with and without the environment infrastructure. Table 10.4 and Table 10.5 resume two excerpts of *Jason* code for

```
+join_req(Ag)
: true
->  make("visitorGroupBoard",
"OMI.GroupBoard",
["moise/hospital.xml","visitGroup"]);
    make("visitorSchBoard",
"OMI.SchemeBoard",
["moise/hospital.xml","visitorSch"]);
    apply("visitorGroupBoard",
adoptRole(Ag, "escort"));
    apply("visitorSchBoard",
commitMission(Ag, "mVisit"));
    include(Ag).

+focus_req(Ag, "BillingMachine")
: true
-> apply("visitorSchBoard",
commitMission(Ag, "mPay")).
```

```
+op_completed("Desk",
    Ag, bookVisit)
: true
-> apply("visitorSchBoard",
    setGoalAchieved(Ag, book_visit)).

+op_completed("BillingMachine",
    Ag, pay)
: true
-> apply("visitorSchBoard",
    setGoalAchieved(Ag, pay_visit)).

+ws_leaved(Ag)
: true
-> apply("visitorGroupBoard",
        leaveRole(Ag, "patient")).
```

Table 10.3: Embodied Organization Rules instrumenting the hospital workspace and supporting the activities of the escort agent.

agents playing the role escort inside the hospital scenario. In this case we assume that both the agents join to a visitor group and commit to the same missions *mVisit* and *mPay*. In so doing, both the agents are assumed to fulfill the same tasks inside the group, in particular by achieving the goals book the visit and pay visit.

10.3.1 Agents at work with Organizational Infrastructure

In Table 10.4 the agent works in a workspace instrumented with an organizational infrastructure and is assumed to interact with organizational artifacts as they have been described in Chapter 8. In this case the agent has in charge the task to create instances of organizational artifacts required for managing its group. Thus, after having joined the hospital workspace, the agent creates a new instance of `visitorGroupBoard` and `visitorSchBoard`. Once those artifacts are in place, the agent starts its activities by interacting with the organizational infrastructure: the `visitorGroupBoard` and `visitorSchBoard` are focused and the role escort is adopted by using the `adoptRole` operation on the `visitorGroupBoard` already created. Now on, the whole agent behavior is governed by the signals perceived from the artifacts.

In particular, an event coming from the scheme board signalling that the role adoption succeeded makes the agent to commit the mission *mVisit*. Besides, an

```
// initial goal
!create_group.

+!create_group
<- cartago.joinWorkspace("Hospital");
   cartago.makeArtifact("visitorGroupBoard",
     "OMI.GroupBoard", ["moise/hospital.xml", "vgroup"], GrId);
   +a_id(gruopBoard, GrId);
   cartago.makeArtifact("visitorSchBoard",
     "OMI.SchemeBoard", ["moise/hospital.xml", "vscheme"], SchId);
   +a_id(schemeBoard, SchId);
   cartago.focus(GrId);
   cartago.focus(SchId);
   cartago.use(GrId, adoptRole(escort)).

+role_adopted(GrId, escort, Ag)
[artifact("visitorSchBoard"), artifact_id(SchId)]
: a_id(schemeBoard, SchId) & .my_name(Ag)
<- cartago.use(SchId, commitMission(mVisit));
   !execute_book.

+new_possible_goal(pay_visit)
[artifact("visitorSchBoard"), artifact_id(SchId)]
: a_id(schemeBoard, SchId)
<- cartago.use(SchId, commitMission(mPay));
   !execute_pay.

/* Purposive Activities */
+!execute_book
<-  // activities related to fulfill bookings
-!execute_book
<- // handle failures

+!execute_pay
<-  // activities related to fulfill payments
-!execute_pay
<- // handle failures

// synchronization: a message is received from the patient agent
+fulfilled(visit)[source(patient)]
<- cartago.leaveWorkspace("Hospital").
```

Table 10.4: *Jason* excerpt for the escort agent working in the hospital workspace instrumented with an Artifact Based OMI

event signalling that the goal pay visit is possible makes the agent to fulfill the payment. Finally, an ACL message received by the patient agent playing inside the same group indicates to the escort that the medical visit has been finalized, thus the agent can leave the workspace.

Although the workflow of activities can be easily expressed in terms of plans reacting to the events signalled by the OAs, the fulfillment of a series of purpo-

```
// initial goal
!enter_room.

+!enter_room
<- cartago.joinWorkspace("Hospital");
   cartago.lookupArtifact("Desk", DeskId);
   +a_id(desk, DeskId);
   cartago.lookupArtifact("BillingMachine", BmId);
   +a_id(billing_machine, BmId);
   cartago.lookupArtifact("Tablet", TabId);
   cartago.focus(TabId);
   !!assist_patient.

+!assist_patient
<- !execute_book;
   !execute_pay;
   cartago.leaveWorkspace("Hospital").

/* Purposive Activities */

+!execute_book
: a_id(desk, DeskId) & booking(Params)
<- cartago.use(DeskId, bookVisit(Params), Ref_Code).

// wait and resume in case of failures
-!execute_book
<- .wait(1000);
   !!execute_book.

+!execute_pay
: a_id(billing_machine, BmId) & payment(Params)
<- cartago.use(BmId, pay(Params), Receipt).

// wait and resume the activity in case of failures
-!execute_pay
<- .wait(1000);
   !!execute_pay.

// synchronization: an event is perceived from the Tablet artifact
+signed(patient)[artifact("Tablet")]
<- cartago.leaveWorkspace("Hospital").
```

Table 10.5: *Jason* excerpt for the escort agent working in the hospital workspace instrumented with an Embodied Organization.

sive activities in this case is in charge of agent's tasks. Notice that the pragmatic actions related to booking and payment are not specified in the provided excerpts. In this case the sole organizational infrastructure does not provide functionalities for these purposes, and the agent has to find the means to achieve these goals autonomously.

10.3.2 Agents at work with Embodied Organization

In Table 10.5, instead, the workspace is instrumented with both organizational and environmental infrastructures (EMI and OMI) as described in Chapter 9. We suppose that the hospital workspace has been programmed using Embodied Organization Rules (i.e., constitutive rules) between the two dimensions. In this case the organization result an embodied infrastructure inside the work environment, being supported by an EMI that can be exploited as an environmental extension of the organizational entity.

Differently from the version specified in Table 10.4, here the agents can prescind form the organizational infrastructures and succeed their activities working directly with environmental artifacts. Hence, after having joined the workspace, the escort agent simply locates the artifacts available in that workspace and suddenly start to interact with them. A set of Embodied Organization Rules works in background to define global dynamics inside the instrumented workspace, and are aimed at make it transparent the functionalities provided by the organizational infrastructure (see Section 9.4). Through the mechanism defined by workspace rules, the Embodied Organization Rules are assumed to transform the events elicited by the agent working with the EMI in the corresponding events affecting the OMI.

In this case a first Embodied Organization Rule is specified stating that joining the workspace "count as" creating the organizational artifacts, adopting the role escort and committing the mission *mVisit* ((Table 10.3, left)) A second rule is then specified indicating that focusing the billing machine "count as" committing the mission *mPay*. Further rules specify that using the desk and the billing machine "count as" achieving the goals book the visit and pay the visit (Table 10.3, right).

Given this configuration, the escort agent does not need to interact with the organization anymore – being the organization automatically acknowledged by the Embodied Organization Rule instrumented inside the workspace. The agent program is thus concretely simplified, while the activities are now related to the achievement of agent's pragmatic goals only (i.e., `execute_book, execute_pay`). In other terms, agents have not more in charge the task to actively manage the organizational entity, which indeed is automatically reconfigured on the basis of the events that actually characterize the workspace.

As an additional benefit, the environment infrastructure also provides agents with the means to achieve goals in practice. Thereby, the goal book the visit can be fulfilled by externalizing an action using the `bookVisit` operation on the desk artifact. Similarly, the pay visit goal is fulfilled using the `pay` operation on the billing machine artifact, and so on. Besides, the possibility to observe artifact

events allows the agent to synchronize its activities with the ones of the patient agent by focusing the tablet artifact, thus without the need to send and receive messages, neither the charge to understand their content. In this case we assume that a tablet signal, indicating that the visit has been finalized by the patient, allows the escort to recognize that it is time to leave the workspace. Notice that to handle possible failures, due to organizational regimentations, the agent here simply uses a couple of plans to be triggered in case of fail events occurring within the artifact use. The result is that the organizational functionalities are made transparent for the agent, while it can be completely unaware of the organizational entity.

10.4 From Situated to Embodied Organizations

The model presented in this chapter proposes the notion of Embodied Organization as a multifaceted artifact based infrastructure deployed in MAS work environment. It mainly differs form other approaches related the notion of situated organizations (a survey of related work is in Chapter 3) for presenting a structured approach applied to both organizational and environmental infrastructures. A series of remarkable aspects and relevant properties are discussed below, with respect to the identified objectives and related works.

10.4.1 Relevant aspects

CONSTITUTIVE REALITIES. As in the approach supported by Normative MAS by Dastani et al. [33, 34], we explicitly model constitutive rules as a mechanism to shift events from environmental to organizational reality, with the aim to maintain a coherent configuration of the global system. In our case, this promotes a seamless integration between two infrastructures, enabling in particular a mediation role to be played by environmental artifacts. *Embodied Organization Rules* are introduced as a programmable layer in order to to control the social behavior and to bridge the gap between the abstract concepts expressed at organizational level, and concrete functionalities characterizing the work environment. As a result, agents can implicitly interact with the organization, even being not aware of its functioning nor of its presence. On the other hands, the organization can enact a direct control upon the environment, thus implementing an effective mechanism for ensuring and regimenting its norms.

The approach of constitutive rules is not followed in Okuyama et al. [91], where the organizational entities and the situated normative objects are not sup-

posed to work together in ensuring norms and monitoring and control of ongoing activities. Besides, the use of objects has a weak effect to the institutional dimension (the inspection of a normative object can not be functionally related to the rest of the organization).

The same approach is adopted in MASQ [132, 6] too, where the laws determining the interference among objects are restricted at defining physical constraints (i.e., objects coming into collision) while the possibility to dynamically specify specific interferences between the physical and the organizational dimensions is not provided.

DECENTRALISED INFRASTRUCTURES. According to the proposed model, an embodied organization realizes an organizational entity inside the global work environment in terms of two decentralised infrastructures based on several type of artifacts. As opposed to alternative views, as the normative model proposed by Dastani et al., the notion of embodied organization envisages environments as constituted by a series of decentralised artifacts, each specified to serve a particular purpose for supporting agents activities.

A similar notion of decentralised environment is followed by the MASQ approach, where agents are capable of sensing data from a series of physical environmental objects. Besides, MASQ objects are assumed to react to the influences performed by agents and possibly propagate these influences according to a set of interference laws specified within the influences and reactions model [51].

The adoption of environmental objects inside the work environment is also promoted by Okuyama et al. Besides providing a notion of locality similar to the one given by A&A workspaces, environment objects assume in this approach the boundaries of normative objects, and can be exploited at the application level by agents themselves for normative regulation. Basically the functions of a normative object resume the ones provided by the observability of organizational artifacts: indeed, normative objects are passive computational entities with an inspectable informative content, which makes available situated information about norms within the place where such objects can be perceived by agents. Similarly to the embodied organization approach, normative objects operate as a coordination media used to convey situated normative content to the agents. Differently from our model, there is not a structured approach in modeling organizational entities, being their organizational specification solely based on the definition of norms in terms of readable rules.

It has to be remarked that both MASQ and Okuyama et al. are mainly addressed at developing a methodological tool and a framework for social simula-

tions, while the aim of our approach is general enough to be addressed to a wider area of software development. As inherited by the A&A model formally described in Chapter 5, our approach deals with decentralised entities. It respects unquestionable principles of programming distributed systems, as synchronizing facts and events (through linking), ensuring mutual exclusion on resources (synchronized access), preventing concurrent executions (only one operation in execution at a time inside the same artifact), etc.

INTERACTIONS. In an embodied organization, each artifact consists in a non autonomous / reactive computational entity, developed with the aim to be suitably exploited by agents at an application level. On these basis, interactions between agents and organizations obey to unambiguous rules defined at a programming level, and are modeled through agent native capabilities of action and perception. Whereas the overall work environment runs on a dedicated platform (i.e. CArtAgO), heterogenous agents can share the same organizational functionalities simply by integrating the repertoire of actions needed to join a workspace and work with artifacts.

For many aspects, the interaction model adopted in A&A- CArtAgO resembles the one proposed by the influences and reaction principles by [51] – recently revised in the context of the MASQ model. The idea at the basis of influences/reactions is that an agent can not directly change the state of the world, an agent decides the action to do and then the environment determines its consequences. An important consequence stressed in that approach is that everything that is not provided by the environment is simply not feasible for an agent.

The approach to interactions envisaged within embodied organizations based on agents and artifacts promotes the notion of mediated interaction. This results as a suitable option for communication between agents (see [106] for an example of agents sharing relevant information mediated by artifacts). Although not detailed in the context of this work, other kinds of interactions can be adopted as well. Most important, agents are not prevented in adopting direct communication based on message exchange and ACLs. There are at least two options to integrate a message based interaction in the proposed model. The first simple option is to use infrastructures already available in agents platforms. Indeed, most of the agent based frameworks already offer embedded services supporting ACL and messaging conforming to FIPA standards (an example is JADE, upon which most agent platforms are based [8]). A less obvious option would be to integrate (and build accordingly) an artifact based infrastructure supporting ACL. This last option, not already pursued in this work, envisages the realization of the whole messag-

ing infrastructure as an artifact based infrastructure. Such an infrastructure may implement, for instance, message boxes as agents' personal artifacts, and global services like white and yellow pages by introducing shared artifacts distributed for each workspace. Addressing the dialogical dimension inside the organization, together with a possible technological support of existing approaches based on dialogical interactions (as the one provided by Electronic Institutions and AMELIE [47]), are actually planned as future work.

FUNCTIONALITIES. Related approaches to situated organization typically refer to a collection of institutional and brute facts, represented as organizational / institutional states and environmental / brute states respectively. For instance, normative objects in Okuyama et al. only provide informational contents to agents, while brute and institutional facts in Dastani et al. only provide a symbolic representation of current stases inside the system. Thereof, these approaches refer to information contents, thus not providing an explicit support to other kinds of functionalities.

In our approach, each artifact provides two kind of functionalities, namely observable properties and operations. These functionalities are then extended to the overall infrastructures in order to be exploited by agents for their *epistemic* (artifacts have an observable state) and *pragmatic* (artifacts have triggerable operations) purposes. This introduces the notion of infrastructural functionality, which in the case of embodied organization can be extended either at the environment level either at the institutional one. Functionalities are made available to agents to interact with the infrastructures, and can be exploited either to modify the brute reality (EMI) either the institutional reality (OMI).

OPENNESS. As emphasized in Chapter 4, openness in artifact based work environments can be intended in a couple of meanings. On the one side, the system is open for agents, thereby artifacts and infrastructures are designed ignoring agents internal models, namely their architectures, their purposes, their effective capabilities. On the other side, the system is open for artifacts, thereby agents entering the work environment may ignore artifacts details at design time, namely which kind of functionalities they provide, how they are located, which (usage) protocols are needed to exploit them, etc. From the point of view of an agent, openness can be faced by the mean of manuals, artifact meta descriptors or any other kind of discovery capability (see Subsection 4.5.2 for an example of this use). From the point of view of the organizational infrastructure, openness can be faced by forcing norms, hence regimenting and enforcing a desired pattern of behavior (as explained in Subsection 2.3.3).

By mapping agent's actions onto artifact operations we have a further important outcome for what concerns openness and dynamism: the repertoire of actions available to the agents is dynamic, it depends on the current shape of the environment (i.e., the set of artifacts actually available in the workspace). In this view, agent's capabilities can be then extended or specialized by agents themselves creating new artifacts or replacing existing ones. Besides, the infrastructures can be updated, replaced, modified, thanks to the operations allowing to dynamically change the functioning (i.e., the program) of existing artifacts at runtime. For instance, by submitting a new instance of NOPL to an OMI, an organizational agent may change the overall organizational specification on the fly. At the same time, the programming model defining Embodied Organization Rules can be easily recasted to relate organizational and environmental infrastructures on situated requirements, while the functional relationships established between the various artifacts promote a comprehensive approach to the whole organization as a unified embodied entity.

DIFFERENT CONCERNS. Emphasizing the separation of concerns between organizational specification and environment specification, artifacts can hold either to the organizational infrastructure, resulting as organizational artifacts, or to the environment infrastructure, resulting as environmental artifacts. The whole work environment is thus unambiguously divided in two separated infrastructures placed at two distinct conceptual levels: an organizational level containing organizational entities detailed with organizational specifications (including norms, roles, collective goals, groups) and an environmental level, containing those environment artifacts to be exploited by agents and monitored / controlled by the organization itself. This, for instance, accounts for maintaining an organizational program focused on organizational constructs only, thus not introducing, at the organizational level, low level details related to the specification of whole apparatus. A similar notion is introduced by the MASQ approach which explicitly deals with four dimensions, ranging from endogenous aspects (related to agent's mental attitudes and cultures) to exogenous aspects (related to environments, society and socio-sphere where agents are immersed). The aim of MASQ is to propose an unique approach by which the various basic elements that compose the interaction processes as agents, environments and organizations are modeled in an integrated way. Similarly to our approach, in MASQ a clear conceptual separation exists between physical (and social) spaces and agents working in it. Differently from our approach, in MASQ there is no explicit representation of an institutional reality external to agent minds. In fact, agents are assumed to learn and internalize insti-

tutional facts in terms of mental attitudes (that they refer to as *cultures*, that are modeled in terms of special beliefs), while the representation of the social reality, as well as the current normative dimension, are always subjectively appraised.

Handling different concerns places an important difference with existing approaches to situated organizations which include the environment as a specific dimension to be programmed *within* the organization [33, 34]. In our approach either organizational entities and environmental ones can be programmed according to their specific needs and they accordingly may evolve as independent (sub) systems. As opposed to the approaches mixing together organizational and environmental specifications, in our approach environment infrastructures can pre-exist the organization, thus supporting and wrapping a series of external services and legacy systems which existence is independent from the organization itself.

(UN)AWARENESS. By mediating between agents and organization, the environment infrastructure give rise to functional, unaware, collective phenomena which can be fully controlled by the organization. As said, the environment infrastructure has a functional role inside the whole system, serving agents as external resources to fulfill their pragmatic and epistemic purposes. This aspect has important consequences at an organizational level.

A first consequence is that, besides organizational agents managing the organization and user agent participating the organization, a new kind of agent may evolve in the system: *unaware agent*. The notion of unawareness refers to agents having no explicit representations nor internal capabilities to reason in organizational/normative terms. They are not assumed to directly exploit an organizational entity, because they have no access to it due to some organizational policy, or due to a lack in their reasoning model, or because of a design choice. In the hospital scenario, for instance, visitors could have no knowledge about the organizational specifications nor they need them to directly exploit the OMI services. They can be, indeed, unaware, and likely not supposed to know the specific protocols needed to participate the organization.

Unawareness is an aspect which other approaches to situated organizations do not address. For instance either in Okuyama et al. either in MASQ, it is explicitly required for agents to retrieve in the interaction spaces those information aimed at informing agents about the institutional rules. Such an information is then internalized through perception, thus allowing agents to bring about it. It has to be remarked that the possibility for agents to work inside an institutional context without being aware of its effective structures facilitates the construction of open systems, where either agents either organizations are assumed to ignore

the counterparts at design time.

A second consequence refer to more complex patterns of interaction, involving agents able to cognitively (rationally) exploit organizational entity through its environment extensions. In fact being able to bring about environmental functionalities may produce implicit effects at an organizational level. Intelligent agents could operate upon environmental infrastructures with the additional aim to alter the organizational configuration, as well as organizational entities may directly operate upon environments so as to signal relevant information to be cognitively appraised by agents. The cognitive model behind such a kind of interaction can be identified, among others, in Castelfranchi theory of Behavioral Implicit Communication (BIC) [24], which assumes agents having mind reading abilities, able to ascribe intentions and to interfere with other agents in terms of mental states. A first step in this direction has been described in [106] and [116], while further investigation is planned for future work.

EVENTS. In the proposed approach the interaction model is entirely conceived in terms of events: events are dispatched by artifacts during their operation execution, events are perceived by agents and events are dispatched due to agent activities (as joining/leaving workspaces, linking/unlinking and creating/diaposing artifacts, etc.). Thereby, differently from related approaches – as for instance the one proposed by Dastani et al. working in terms of physical and institutional *facts* – the proposed approach allows to deal with relevant changes occurring in the environments in terms of workspace *events*, which can be suddenly interpreted in terms of relevant changes in the application domain. A coherent set of mechanisms is then available to trace a rich taxonomy of events and handle them ether at the subjective level of agents, either at the infrastructural level by programming intra-workspace dynamics. Indeed, events are then possibly perceived by agents or intercepted by workspace rules. Basically this enables developers (and agents themselves) to face environments dynamics at a finer granularity, effectively improving situatedness of the whole system. As a consequence, the proposed approach allows to deal with the notion of "attempt" for agents who tries to execute artifact operations. The outcome of a given operation can be indeterministic and it may be established taking int account the actual configuration of the infrastructure (i.e. the institutional facts stored inside the organizational artifacts by NOPL constructs).

10.4.2 Limitations and drawbacks

Of course the proposed approach is not free from drawbacks. A first issue concerns the processing model adopted to implement Embodied Organization Rules. Using Embodied Organization Rules requires a rule engine to be managed inside each workspace. Besides, the mechanism of workspace rules, on top of which Embodied Organization Rules are provided, assumes a series of strict assumptions which may elicit a considerable computational load once complex rules are run. As showed in Chapter 6, each rule requires atomic transitions with the execution of multiple operators. It also requires to resume a consistent configuration of the whole workspace once some failure occurs. Indeed, a successful reaction can atomically modify the configuration of the workspace, while a failed reaction yelds no changes at all. These aspects introduce further computational load within the rule engine, and may suggest the developer to minimize the use of workspace rules in programming artifact based infrastructures. Notwithstanding, the need to provide a comprehensive approach to Embodied Organizations goes in the opposite direction: minimize the effort for agents in interacting with the organizational entities, thus delegating as much as possible to Embodied Organization Rules.

An additional limitation concerns programming complex organizational patterns: it may result hard to specify due to the functional level at which the Embodied Organization Rules actually operate (i.e. single artifact functions).

The above mentioned aspects are partially soften by the distribution model which accounts the possibility to distribute the system over multiple nodes and workspaces—Embodied Organization must be deployed in a single workspace to guarantee a cohesive unity of time and space. This origins a methodological trade-off, by which the developer has to decide which part of the interaction could be managed with Embodied Organization Rules, and which part has to be left in charge of participating agents. At the moment, we argue that in order to overcome these limitations a certain amount of experience in building concrete Embodied Organization is needed. Although it may be a downside due to the novelty of the approach, of course future work will be addressed in better understanding this kind of issues.

10.5 Final Remarks on Programming Embodied Organizations

As highlighted by the concrete description of the programming model provided in this chapter, the presence of an environmental infrastructure accompanying the organizational one has a series of remarkable benefits, for instance in terms of separation of concerns and decentralization, rich event based interactions, functionalities, openness, agent awareness, etc. As a main outcome, agents can operate in a goal oriented fashion directly upon environment artifacts, thereby exploiting entities embodied in their work environment for their purposes, without explicitly take into account the organizational entities, nor the additional activities needed to inform the organization about their ongoing activities. As far as the model has been conceived, the organization result as an "hidden" infrastructure, transparently deployed behind the environment artifacts. This results in the possibility for the organizational entity to be automatically acknowledged about agent's ongoing activities, thanks to the presence of embodiment rules regulating the functional links between environmental and organizational infrastructures. Furthermore, the same embodiment rules allow the organization to operate a functional control over environmental resources, thus implementing regimentation and enforcement over agents.

Resuming the abstract picture of interactions discussed in the introduction of this thesis (Figure 1.1), the proposed approach defines a series of mechanisms placed at an "agentive" level of abstraction: they are aimed at better situating agents, environment and organization in the context of an integrated framework. In this perspective, interactions taking place between agents and environment resources (A-E relationships) have been shaped on a semantic typical of agent capabilities, as the one of action and (event) perception. Besides, interaction between environments and organizations (E-O relationships) have been conceived in terms of *constitutive rules*, and realized by the event based mechanism of workspace rules allowing the specification of global laws inside the environment. Finally, interactions between organizations and agents (O-A relationships) have been provided with the mediation role played both by agents and artifacts, thus exploiting the notion of regimentation and organizational agents.

Chapter 11

Conclusions

This thesis deals with programming organizations in agent systems. It devises in particular a computational model for organizational entities situated in structured agent work environments. To this end, three main perspectives related to agents, environments and organizations are considered in order to provide an integrated approach. The resulting programming model – and the related technology – are aimed at put in relation all the different entities characterizing the system by providing a global, coherent view. In this perspective, the notion of *Embodied Organization* is introduced relating to the particular configuration of a Multiagent System instrumented with a twofold infrastructure, concerning both organizational and environmental aspects.

As recognized for instance by [141] software development over the last decades has been characterized by an evolution from a mathematics-centered model focused on data processing, to one based on *interaction* among distributed and heterogenous entities. This evolution continues today not only through the advances in hardware platforms but mostly thanks to the continuous refinements of the abstraction tools adopted in the development of ever more complex programs. The same kind of evolution could be encountered also in this work, where a main research objective has been addressed at establishing a wide set of interaction styles, which are assumed to take place between the abstraction tools placed by agents, environments and organizational entities. Interactions have been conceived without loosing the objectives of programming the system as a coherent ensemble. A particular attention has been devised for the abstraction level at which the system can be programmed, with the aim to maintain an “agentive” level of abstraction and not constrain developers to draw systems thinking in terms of “mechanisms”.

This, we argue, results in a first step toward the challenge of reconciling – both at design and at a programming level – emergence with cognition, intentional behavior with social functions, organization aware models of agency with unaware styles of cooperation/coordination.

11.1 Contribution of this Thesis

In carrying out the objectives identified in the first chapters, this work has allowed to finalize a number of research contributions in the field of agent systems. These contributions can be analyzed under the different perspectives upon which the thesis has been structured.

Environment Programming

In the first part of the thesis, a programming model, and a related technology, for implementing infrastructures in Multiagent Systems has been introduced. The approach relies on research on environment programming promoted within the A&A (Agents & Artifacts) model, where artifacts and workspaces are assumed as basic entities structuring decentralised work environments. MAS infrastructures are indeed based on the notion of artifact and workspace, provided as the basic computational units promoting a structured programming model of MAS environments. In this view, agents are autonomous, proactive entities of the system, and are assumed to dynamically use, replace, change artifacts (and thus, infrastructures) based on their actual needs. On the other hands, artifacts are considered passive, non autonomous, distributed entities structuring the work environment in terms of functionalities and services to be exploited by agents.

CArtAgO has been proposed as a concrete technology for implementing infrastructures in practice, based on the A&A model. Its concrete programming model has been developed including a series of contributes by the author of this thesis. In Chapter 4, the CArtAgO programming platform has been detailed, also describing the insights of a series of concrete use cases where agents externalize their tasks inside work environments instrumented with artifacts. The interaction model has been defined either at a theoretical and at a programming level, as based on agentive capabilities of actions and perceptions. In this view, heterogeneous agent models and architectures are assumed to work inside CArtAgO environments by internalizing a basic set of actions in their repertoire. A main contribution to the framework has been to shape artifacts computational model in

order to match the main features typical of a strong notion of agency, as they can be based on epistemic and motivational attitudes. This research line has allowed agents to exploit artifacts in a coherent interaction style, according to a series of epistemic and pragmatic functionalities which artifacts provide in terms of observable properties and usage interfaces. This resulted in a further contribution on agents programming, related in particular to the definition of those goal oriented capabilities allowing agents to externalize activities on artifact resources. Interoperability and openness are enabled not only at a mechanism level, but also promoting artifact mediation role, which is aimed at enabling, interceding, governing and improving agents interactions and cooperation inside the application domain.

A main contribution to environment programming has been proposed in order to deal with a wide series of interaction styles inside the system. This purpose has suggested the introduction of an additional coordination mechanism, which is a further contribution of this thesis. To this end, a renewed model of artifact based environments has been formally described including its dynamics, and detailing in particular the interactions between agents, artifacts and workspaces as centered on the notion of events (Chapter 5). The pivotal role of events has led to the definition of programming constructs aimed at specifying intra-workspace dynamics besides agent artifact interactions, that is, through a programming language specifying a series of event based rules applying sequences of operators affecting entities actually dwelling the workspace. The execution of such rules is aimed at eliciting additional outcomes once specific events occur inside the workspace, thus altering the normal course of events and providing a suitable approach in specifying global laws inside the system (Chapter 6).

Organization Programming

The programming model used to conceive organizations has been rooted on the $\mathcal{M}$oise model. The approach allows to realize organizations of agents as structured in multiple conceptual dimensions and promotes the definition of a normative specification to be handled in concrete organizational entities deployed inside the MAS. Chapter 7 implements a concrete organizational specification according to the $\mathcal{M}$oise model, thus providing the basis for a concrete scenario (referred as hospital scenario) used as a guideline in the rest of the thesis. By adopting the same approach used for programming environments, organizational entities have been straightforwardly realized as concrete infrastructures based on A&A principles and realized in CArtAgO (Chapter 8). This results in decentralized Organiza-

tional Management Infrastructures (OMI), based on artifacts and workspaces and rooted in the computational model described in the first part of the thesis, with a particular emphasis to the role played by events.

Following the approach recently proposed by Hübner et al., the resulting artifact based OMI is then instrumented by an internal mechanism aimed at entailing both actual organizational configuration and the dynamic constraints specified for ensuring its global coherence. Such a mechanism devises the constructs of a normative programming language (NOPL) obtained from an automatic translation of the $\mathcal{M}$oise specification. As emphasized, the functionalities provided by the individual artifacts have been shifted to an infrastructural level, thus introducing the notion of organizationa/institutional facts (i.e., in terms of artifact observable properties), as well as the notion organizational/institutional functions (i.e., in terms of artifact usage interfaces).

Unifying approach to Programming Organizations, Agents and Environments

In the last part of the thesis, the notion of *Embodied Organization* has been introduced by considering an additional environment support to be provided aside the one provided by the organizational infrastructure. A complementary infrastructure (referred as Environment Management Infrastructure - EMI) has been described as an environmental extension of the organizational entity, thus providing agents with additional functionalities to be exploited at the application level.

In Chapter 9, Embodied organizations have been introduced as a unified approach to organizational and environmental infrastructures, promoting an integrated approach to programming different dimensions as a coherent ensemble. In order to establish functional relationships between the heterogenous entities at the basis of organizations and environments, special programmable constructs have been defined, inspired by the notion of constitutive rules introduced in social sciences by John Searle. These principles have been rooted on the event based mechanism regulating global dynamics inside workspaces, and realized in terms of *Embodied Organization Rules*. The programming model at the basis of Embodied Organization Rules has been implemented with the same mechanisms used for regulative rules specifying intra-workspace dynamics.

Finally, based on the guideline scenario previously introduced, Chapter 10 provides a concrete description of how embodied organization can be engineered in practice. The description is enriched by a series of concrete examples aimed at providing a practical methodology and guidance to developers.

11.2 Future directions

Future work will account further investigation about the notion of embodied organizations and will be addressed at covering missing aspects, such as the dialogical (and normative) dimension of interactions. An important aspect actually deserving our attention is the computational model for agents able to reason in organizational terms, with particular emphasis on the cognitive mediators needed for agents to be aware of organizational constructs. Besides, a challenging direction accounts the possibility to integrate human agents in the proposed approach, thus exploring real life scenery and refining the notion of “embodiment” with the realization of sociotechnical systems.

Among the ongoing activities, the main one is the distribution of a new release of CArtAgO integrating both artifact based environments and organizations regulated by embodied mechanisms. In this line, a particular attention will be devised for easing the work of developers, i.e., by enriching the release with support material and development tools.

Finally, an important long term objective would be the definition of a general purpose approach, towards the full integration of the proposed model in the context of mainstream agent oriented programming.

Appendix A

Moise specification for the Hospital Scenario

This appendix includes the complete $\mathcal{M}$oise OML specification for the organizational entity referred in the context of the hospital scenario. This specification is expressed in XML and can be straightforwardly desumed from the graphical specification described in Chapter 7.

```
<?xml version="1.0" encoding="UTF-8"?>

<?xml-stylesheet href="http://moise.sourceforge.net/xml/os.xsl" type="text/xsl" ?>

<organisational-specification id="hospitalspec"
            os-version="0.7"
            xsi:schemaLocation="http://moise.sourceforge.net/os
                       http://moise.sourceforge.net/xml/os.xsd"
            xmlns="http://moise.sourceforge.net/os"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

            <!-- STRUCTURAL SPECIFICATION-->
            <structural-specification>
                <role-definitions>
                    <role id="staff"/>
                    <role id="doctor">
                      <extends role="staff"/>
                    </role>
                    <role id="visitor"/>
                    <role id="escort">
                      <extends role="visitor"/>
                    </role>
                    <role id="patient">
                      <extends role="visitor"/>
                    </role>
                </role-definitions>

                <group-specification id="surgery_room_group">
                       <links>
                           <link from="staff" to="visitor" type="communication"
                              scope="inter-group" extends-sub-groups="true" bi-dir="true"/>
                       </links>
                       <sub-groups>
                            <group-specification id="visit_group" max="100" min="0" monitoring-scheme="monitoringSch">
                                <roles>
```

```
                        <role id="escort" min="0" max="1"/>
                        <role id="patient" min="1" max="1"/>
                    </roles>
                    <links>
                        <link from="patient" to="escort" type="acquaintance" scope="intra-group"
                                 extends-sub-groups="true" bi-dir="false"/>
                        <link from="patient" to="escort" type="communication" scope="intra-group"
                                 extends-sub-groups="true" bi-dir="true"/>
                    </links>
                    <formation-constraints>
                        <compatibility from="patient" to="escort" type="compatibility" scope="intra-group"
                                 extends-sub-groups="false" bi-dir="false"/>
                    </formation-constraints>
                </group-specification>

                <group-specification id="staff_group" max="1" min="1" monitoring-scheme="monitoringSch">
                    <roles>
                        <role id="doctor" min="1" max="1"/>
                        <role id="staff" min="0" max="1"/>
                    </roles>
                    <links>
                        <link from="doctor" to="staff" type="acquaintance" scope="intra-group"
                                extends-sub-groups="true" bi-dir="false"/>
                        <link from="doctor" to="staff" type="communication" scope="intra-group"
                                extends-sub-groups="true" bi-dir="false"/>
                    </links>
                    <formation-constraints>
                        <compatibility from="doctor" to="staff" type="compatibility" scope="intra-group"
                                extends-sub-groups="false" bi-dir="false"/>
                    </formation-constraints>
                </group-specification>
            </sub-groups>
            <!--<formation-constraints>
              <cardinality min="1" max="1" object="role" id="doctor"/>
            </formation-constraints>-->
    </group-specification>
</structural-specification>


<!-- FUNCTIONAL SPECIFICATION-->
<functional-specification>
   <scheme id="visitorSch">
     <goal id="visitor">
        <plan operator="sequence">
            <goal id="enter" ds="enter the room"/>
            <goal id="book" ds="book the visit"/>
            <goal id="visit" ds="visit">
                <plan operator="parallel">
                    <goal id="do_visit" ttf="30 min" ds="do the visit"/>
                    <goal id="pay_visit" ttf="30 min" ds="pay the visit"/>
                </plan>
            </goal>
            <goal id="exit" ds="exit the room"/>
        </plan>

    </goal>
    <mission id="mPatient" min="1" max="1">
        <goal id="do_visit"/>
    </mission>
    <mission id="mPay" min="1" max="1">
        <goal id="pay_visit"/>
    </mission>
    <mission id="mVisit" min="1" max="2">
        <goal id="enter"/>
        <goal id="book"/>
        <goal id="exit"/>
    </mission>
</scheme>
<scheme id="monitorSch">
    <goal id="monitor">
        <plan operator="sequence">
            <goal id="observe" ds="monitoring agents"/>
            <goal id="enforcement" ds="Reward_Punish some agents">
                <plan operator="choice">
                    <goal id="send_bill" ttf="1 day" ds="reward good visitors"/>
                    <goal id="send_fee" ttf="1 day" ds="punish bad visitors"/>
                </plan>
```

```
                </goal>
            </plan>
        </goal>
        <mission id="mStaff" min="1" max="1">
            <goal id="observe"/>
        </mission>
        <mission id="mRew" min="1" max="1">
            <goal id="send_bill"/>
        </mission>
        <mission id="mSan" min="1" max="1">
            <goal id="send_fee"/>
        </mission>
    </scheme>
    <scheme id="docSch"><!--<goal id="doctor">-->
        <goal id="visit_patient" ttf="30 min" ds="do the visit"/><!-- </goal> -->
            <mission id="mDoc" min="1" max="1">
                <goal id="visit_patient"/>
            </mission>
    </scheme>
    </functional-specification>


    <!-- DEONTIC SPECIFICATION-->
    <normative-specification>
        <norm id="n1"
            type="obligation"
            role="escort"
            mission="mVisit"/>
        <norm id="n2"
            type="obligation"
            role="patient"
            mission="mVisit"/>
        <norm id="n3"
            type="obligation"
            role="patient"
            mission="mPatient"/>
        <norm id="n4"
            type="permission"
            role="escort"
            mission="mPay"
            time-constraint="5 min"/>
        <norm id="n5"
            type="obligation"
            condition="unfulfilled(n4)"
            role="patient"
            mission="mPay"
            time-constraint="5 min"/>
        <norm id="n6"
            type="obligation"
            role="staff"
            mission="mStaff"/>
        <norm id="n7"
            type="obligation"
            role="doctor"
            mission="mDoc"/>
        <norm id="n8"
            type="obligation"
            condition="unfulfilled(n4) | unfulfilled(n5)"
            role="staff"
            mission="mSan"
            time-constraint="1 day"/>
        <norm id="n9"
            type="obligation"
            condition="unfulfilled(n4) &amp; unfulfilled(n5)"
            role="staff"
            mission="mSan"
            time-constraint="1 day"/>
        <norm id="n10"
            type="obligation"
            condition="unfulfilled(n6)"
            role="doctor"
            mission="mStaff"/>
  </normative-specification>

</organisational-specification>
```

Appendix B

NOPL specification for the Hospital Scenario

This appendix contains the source code for the NOPL program specifying the hospital scenario. The program is generated since the Moise specification showed in the previous appendix. A tool actually provided in the Moise platform[1] allows to automatically create the NOPL specification on the basis of the translation rules described in Chapter 7 and in [61, 62].

```
/*
    This program was automatically generated from
    the organisation specification 'hospitalspec'
    on febbraio 24, 2010 - 00:55:10

    This is a MOISE tool, see more at http://moise.sourceforge.net

*/

scope organisation(hospitalspec) {


   // Role hirarchy
   subrole(staff,soc).
   subrole(doctor,staff).
   subrole(visitor,soc).
   subrole(patient,visitor).
   subrole(escort,visitor).
   fplays(A,R,G) :- plays(A,R,G).
   fplays(A,R,G) :- subrole(R1,R) & fplays(A,R1,G).


scope group(surgery_room_group) {

   // ** Facts from OS


   // ** Rules
   rplayers(R,G,V) :- .count(plays(_,R,G),V).
   well_formed(G).
```

[1] Moise is an open source project at: http://moise.sourceforge.net/

```
   // ** Properties check
   norm role_in_group:
           plays(Agt,R,Gr) &
           not role_cardinality(R,_,_)
        -> fail(role_in_group(Agt,R,Gr)).
   norm role_cardinality:
           group_id(Gr) &
           role_cardinality(R,_,RMax) &
           rplayers(R,Gr,RP) &
           RP > RMax
        -> fail(role_cardinality(R,Gr,RP,RMax)).
   norm role_compatibility:
           plays(Agt,R1,Gr) &
           plays(Agt,R2,Gr) &
           R1 < R2 &
           not compatible(R1,R2)
        -> fail(role_compatibility(R1,R2,Gr)).
   norm well_formed_responsible:
           responsible(Gr,S) &
           not monitor_scheme(S) &
           not well_formed(Gr)
        -> fail(well_formed_responsible(Gr)).


   // ** Sub-Group staff_group
scope group(staff_group) {

   // ** Facts from OS
   role_cardinality(staff,0,1).
   role_cardinality(doctor,1,1).

   compatible(doctor,staff).

   // ** Rules
   rplayers(R,G,V) :- .count(plays(_,R,G),V).
   well_formed(G) :-
      rplayers(staff,G,Vstaff) & Vstaff >= 0 & Vstaff <= 1 &
      rplayers(doctor,G,Vdoctor) & Vdoctor >= 1 & Vdoctor <= 1.

   // ** Properties check
   norm role_in_group:
           plays(Agt,R,Gr) &
           not role_cardinality(R,_,_)
        -> fail(role_in_group(Agt,R,Gr)).
   norm role_cardinality:
           group_id(Gr) &
           role_cardinality(R,_,RMax) &
           rplayers(R,Gr,RP) &
           RP > RMax
        -> fail(role_cardinality(R,Gr,RP,RMax)).
   norm role_compatibility:
           plays(Agt,R1,Gr) &
           plays(Agt,R2,Gr) &
           R1 < R2 &
           not compatible(R1,R2)
        -> fail(role_compatibility(R1,R2,Gr)).
   norm well_formed_responsible:
           responsible(Gr,S) &
           not monitor_scheme(S) &
           not well_formed(Gr)
        -> fail(well_formed_responsible(Gr)).
} // end of group staff_group


   // ** Sub-Group visit_group
scope group(visit_group) {

   // ** Facts from OS
   role_cardinality(patient,1,1).
   role_cardinality(escort,0,1).

   compatible(patient,escort).

   // ** Rules
   rplayers(R,G,V) :- .count(plays(_,R,G),V).
```

```
   well_formed(G) :-
      rplayers(patient,G,Vpatient) & Vpatient >= 1 & Vpatient <= 1 &
      rplayers(escort,G,Vescort) & Vescort >= 0 & Vescort <= 1.

   // ** Properties check
   norm role_in_group:
           plays(Agt,R,Gr) &
           not role_cardinality(R,_,_)
        -> fail(role_in_group(Agt,R,Gr)).
   norm role_cardinality:
           group_id(Gr) &
           role_cardinality(R,_,RMax) &
           rplayers(R,Gr,RP) &
           RP > RMax
        -> fail(role_cardinality(R,Gr,RP,RMax)).
   norm role_compatibility:
           plays(Agt,R1,Gr) &
           plays(Agt,R2,Gr) &
           R1 < R2 &
           not compatible(R1,R2)
        -> fail(role_compatibility(R1,R2,Gr)).
   norm well_formed_responsible:
           responsible(Gr,S) &
           not monitor_scheme(S) &
           not well_formed(Gr)
        -> fail(well_formed_responsible(Gr)).
} // end of group visit_group

} // end of group surgery_room_group

scope scheme(docSch) {

   // ** Facts from OS
   mission_cardinality(mDoc,1,1).

   mission_role(mDoc,doctor).

   goal(mDoc,visit_patient,[],achievement,all,`30 min`).

   // ** Rules
   mplayers(M,S,V) :- .count(committed(_,M,S),V).
   well_formed(S) :-
      mplayers(mDoc,S,VmDoc) & VmDoc >= 1 & VmDoc <= 1.

   // conditions for satisfiability
   satisfied(S,G) :-     // no agents have to achieve -- automatically satisfied by its pre-conditions
      goal(_,G,PCG,_,0,_) & all_satisfied(S,PCG).
   satisfied(S,G) :-     // all committed agents have to achieve
      goal(M,G,_,_,all,_) & mplayers(M,S,V)  & .count( achieved(S,G,A), V ).
   satisfied(S,G) :-     // some agents have to achieve
      goal(_,G,_,_,X,_)  & X > 0 & .count( achieved(S,G,A), X ).

   // permitted goals (dependence between goals)
   ready(S,G) :- goal(_, G, PCG, _, NP, _) & NP \== 0 & all_satisfied(S,PCG).
   all_satisfied(_,[]).
   all_satisfied(S,[G|T]) :- satisfied(S,G) & all_satisfied(S,T).


   // ** Norms

   norm n7:
           scheme_id(S) & responsible(Gr,S) &
           mplayers(mDoc,S,V) & V < 1 &
           fplays(A,doctor,Gr)
        -> obligation(A,n7,committed(A,mDoc,S), `now`).

   // --- Goals ---
   // agents are obliged to fulfil their ready goals
   norm ngoal:
           committed(A,M,S) & goal(M,G,_,achievement,_,D) &
           well_formed(S) & ready(S,G)
        -> obligation(A,ngoal(S,M,G),achieved(S,G,A),`now` + D).

   // --- Properties check ---
   norm goal_non_compliance:
           obligation(Agt,ngoal(S,M,G),Obj,TTF) &
           not Obj &
```

```
          `now` > TTF
       -> fail(goal_non_compliance(obligation(Agt,ngoal(S,M,G),Obj,TTF))).
   norm mission_permission:
          committed(Agt,M,S) &
          not (mission_role(M,R) &
          responsible(Gr,S) &
          fplays(Agt,R,Gr))
       -> fail(mission_permission(Agt,M,S)).
   norm mission_cardinality:
          scheme_id(S) &
          mission_cardinality(M,_,MMax) &
          mplayers(M,S,MP) &
          MP > MMax
       -> fail(mission_cardinality(M,S,MP,MMax)).
   norm ach_not_ready_goal:
          achieved(S,G,Agt) &
          not ready(S,G)
       -> fail(ach_not_ready_goal(S,G,Agt)).
   norm ach_not_committed_goal:
          achieved(S,G,Agt) &
          goal(M,G,_,_,_,_) &
          not committed(Agt,M,S)
       -> fail(ach_not_committed_goal(S,G,A)).
} // end of scheme docSch

scope scheme(monitorSch) {

   // ** Facts from OS
   mission_cardinality(mSan,1,1).
   mission_cardinality(mStaff,1,1).
   mission_cardinality(mRew,1,1).

   mission_role(mStaff,doctor).
   mission_role(mSan,staff).
   mission_role(mStaff,staff).

   goal(mRew,send_bill,[observe],achievement,all,`1 day`).
   goal(nomission,enforcement,[],achievement,0,`0 seconds`).
   goal(nomission,monitor,[enforcement],achievement,0,`0 seconds`).
   goal(mStaff,observe,[],achievement,all,`0 seconds`).
   goal(mSan,send_fee,[observe],achievement,all,`1 day`).

   // ** Rules
   mplayers(M,S,V) :- .count(committed(_,M,S),V).
   well_formed(S) :-
      mplayers(mSan,S,VmSan) & VmSan >= 1 & VmSan <= 1 &
      mplayers(mStaff,S,VmStaff) & VmStaff >= 1 & VmStaff <= 1 &
      mplayers(mRew,S,VmRew) & VmRew >= 1 & VmRew <= 1.

   // conditions for satisfiability
   satisfied(S,G) :-     // no agents have to achieve -- automatically satisfied by its pre-conditions
      goal(_,G,PCG,_,0,_) & all_satisfied(S,PCG).
   satisfied(S,G) :-     // all committed agents have to achieve
      goal(M,G,_,_,all,_) & mplayers(M,S,V)  & .count( achieved(S,G,A), V ).
   satisfied(S,G) :-     // some agents have to achieve
      goal(_,G,_,_,X,_)  & X > 0 & .count( achieved(S,G,A), X ).

   // permitted goals (dependence between goals)
   ready(S,G) :- goal(_, G, PCG, _, NP, _) & NP \== 0 & all_satisfied(S,PCG).
   all_satisfied(_,[]).
   all_satisfied(S,[G|T]) :- satisfied(S,G) & all_satisfied(S,T).


   // ** Norms

   norm n10:
          unfulfilled(n6) &
          scheme_id(S) & responsible(Gr,S) &
          mplayers(mStaff,S,V) & V < 1 &
          fplays(A,doctor,Gr)
       -> obligation(A,n10,committed(A,mStaff,S), `now`).
   norm n9:
          unfulfilled(n4) & unfulfilled(n5) &
          scheme_id(S) & responsible(Gr,S) &
          mplayers(mSan,S,V) & V < 1 &
          fplays(A,staff,Gr)
       -> obligation(A,n9,committed(A,mSan,S), `now`+`1 day`).
```

```
   norm n8:
           unfulfilled(n4) | unfulfilled(n5) &
           scheme_id(S) & responsible(Gr,S) &
           mplayers(mSan,S,V) & V < 1 &
           fplays(A,staff,Gr)
        -> obligation(A,n8,committed(A,mSan,S), `now`+`1 day`).
   norm n6:
           scheme_id(S) & responsible(Gr,S) &
           mplayers(mStaff,S,V) & V < 1 &
           fplays(A,staff,Gr)
        -> obligation(A,n6,committed(A,mStaff,S), `now`).

   // --- Goals ---
   // agents are obliged to fulfil their ready goals
   norm ngoal:
           committed(A,M,S) & goal(M,G,_,achievement,_,D) &
           well_formed(S) & ready(S,G)
        -> obligation(A,ngoal(S,M,G),achieved(S,G,A),`now` + D).

   // --- Properties check ---
   norm goal_non_compliance:
           obligation(Agt,ngoal(S,M,G),Obj,TTF) &
           not Obj &
           `now` > TTF
        -> fail(goal_non_compliance(obligation(Agt,ngoal(S,M,G),Obj,TTF))).
   norm mission_permission:
           committed(Agt,M,S) &
           not (mission_role(M,R) &
           responsible(Gr,S) &
           fplays(Agt,R,Gr))
        -> fail(mission_permission(Agt,M,S)).
   norm mission_cardinality:
           scheme_id(S) &
           mission_cardinality(M,_,MMax) &
           mplayers(M,S,MP) &
           MP > MMax
        -> fail(mission_cardinality(M,S,MP,MMax)).
   norm ach_not_ready_goal:
           achieved(S,G,Agt) &
           not ready(S,G)
        -> fail(ach_not_ready_goal(S,G,Agt)).
   norm ach_not_committed_goal:
           achieved(S,G,Agt) &
           goal(M,G,_,_,_,_) &
           not committed(Agt,M,S)
        -> fail(ach_not_committed_goal(S,G,A)).
} // end of scheme monitorSch

scope scheme(visitorSch) {

   // ** Facts from OS
   mission_cardinality(mVisit,1,2).
   mission_cardinality(mPatient,1,1).
   mission_cardinality(mPay,1,1).

   mission_role(mVisit,escort).
   mission_role(mPay,patient).
   mission_role(mPay,escort).
   mission_role(mPatient,patient).
   mission_role(mVisit,patient).

   goal(mPay,pay_visit,[book],achievement,all,`30 min`).
   goal(mPatient,do_visit,[book],achievement,all,`30 min`).
   goal(mVisit,book,[enter],achievement,all,`0 seconds`).
   goal(nomission,visit,[do_visit, pay_visit],achievement,0,`0 seconds`).
   goal(mVisit,enter,[],achievement,all,`0 seconds`).
   goal(nomission,visitor,[exit],achievement,0,`0 seconds`).
   goal(mVisit,exit,[visit],achievement,all,`0 seconds`).

   // ** Rules
   mplayers(M,S,V) :- .count(committed(_,M,S),V).
   well_formed(S) :-
      mplayers(mVisit,S,VmVisit) & VmVisit >= 1 & VmVisit <= 2 &
      mplayers(mPatient,S,VmPatient) & VmPatient >= 1 & VmPatient <= 1 &
      mplayers(mPay,S,VmPay) & VmPay >= 1 & VmPay <= 1.

   // conditions for satisfiability
```

```
   satisfied(S,G) :-     // no agents have to achieve -- automatically satisfied by its pre-conditions
      goal(_,G,PCG,_,0,_) & all_satisfied(S,PCG).
   satisfied(S,G) :-     // all committed agents have to achieve
      goal(M,G,_,_,all,_) & mplayers(M,S,V)  & .count( achieved(S,G,A), V ).
   satisfied(S,G) :-     // some agents have to achieve
      goal(_,G,_,_,X,_)  & X > 0 & .count( achieved(S,G,A), X ).

   // permitted goals (dependence between goals)
   ready(S,G) :- goal(_, G, PCG, _, NP, _) & NP \== 0 & all_satisfied(S,PCG).
   all_satisfied(_,[]).
   all_satisfied(S,[G|T]) :- satisfied(S,G) & all_satisfied(S,T).


   // ** Norms

   norm n1:
           scheme_id(S) & responsible(Gr,S) &
           mplayers(mVisit,S,V) & V < 2 &
           fplays(A,escort,Gr)
        -> obligation(A,n1,committed(A,mVisit,S), `now`).
   norm n5:
           unfulfilled(n4) &
           scheme_id(S) & responsible(Gr,S) &
           mplayers(mPay,S,V) & V < 1 &
           fplays(A,patient,Gr)
        -> obligation(A,n5,committed(A,mPay,S), `now`+`5 min`).
   norm n3:
           scheme_id(S) & responsible(Gr,S) &
           mplayers(mPatient,S,V) & V < 1 &
           fplays(A,patient,Gr)
        -> obligation(A,n3,committed(A,mPatient,S), `now`).
   norm n2:
           scheme_id(S) & responsible(Gr,S) &
           mplayers(mVisit,S,V) & V < 2 &
           fplays(A,patient,Gr)
        -> obligation(A,n2,committed(A,mVisit,S), `now`).

   // --- Goals ---
   // agents are obliged to fulfil their ready goals
   norm ngoal:
           committed(A,M,S) & goal(M,G,_,achievement,_,D) &
           well_formed(S) & ready(S,G)
        -> obligation(A,ngoal(S,M,G),achieved(S,G,A),`now` + D).

   // --- Properties check ---
   norm goal_non_compliance:
           obligation(Agt,ngoal(S,M,G),Obj,TTF) &
           not Obj &
           `now` > TTF
        -> fail(goal_non_compliance(obligation(Agt,ngoal(S,M,G),Obj,TTF))).
   norm mission_permission:
           committed(Agt,M,S) &
           not (mission_role(M,R) &
           responsible(Gr,S) &
           fplays(Agt,R,Gr))
        -> fail(mission_permission(Agt,M,S)).
   norm mission_cardinality:
           scheme_id(S) &
           mission_cardinality(M,_,MMax) &
           mplayers(M,S,MP) &
           MP > MMax
        -> fail(mission_cardinality(M,S,MP,MMax)).
   norm ach_not_ready_goal:
           achieved(S,G,Agt) &
           not ready(S,G)
        -> fail(ach_not_ready_goal(S,G,Agt)).
   norm ach_not_committed_goal:
           achieved(S,G,Agt) &
           goal(M,G,_,_,_,_) &
           not committed(Agt,M,S)
        -> fail(ach_not_committed_goal(S,G,A)).
} // end of scheme visitorSch

} // end of organisation hospitalspec
```

Index

Action
  in CArtAgO, 68, 78
    alignment, 70
    tag, 70
    timeout, 70
  model, 47
Activity Theory, 53
Agent Centered MAS (ACMAS), 17
Agent Communication Languages (ACL), 21
Agent Coordination Context (ACC), 130
Agent Oriented Programming, 1
Agents, 55
  Cognitive, 82
  configuration, 96
  Organizational, 22, 33, 181, 197, 208
  Programming, 79
  Staff, *see* Organizational
  working with Embodied Organization, 212
  working with OMI, 209
Agents & Artifacts (A&A), 53–59, 96
AGR, 18
AMELIE, 21, 42
Artifacts, 55, 64
  configuration, 97
  Coordination, 146
  Environmental, 192
  Group, 178
  Group board, 178
  Linkability, 67, 72
  Manual, 68, 71, 88, 196
  Operations, 66
  Organizational, 176
  Properties, 56
  Scheme, 177
  Scheme board, 177
  Signals, 67
  Type, 66
Aspect Oriented Programming, 146

Behavioral Implicit Communication (BIC), 219
Beliefs, 86

CArtAgO, 65–79
  Artifact type, 73
  Operation, 74
    feedback, 76
    guard, 76
    internal, 76
    multi step, 74
Constitutive rules, 41, 197, 204–208
Coordination
  Media, 57
  Support, 58
Coordination Media, 146

Electronic Institutions, 21, 182
Emb-Org-Rule, 200–201, 204–208
Environment

centralized approaches, 49
computational model, 49
data model, 52
decentralized approaches, 50
distribution model, 52
dynamics, 51
in AI, 44
in AOSE, 45
structured approach, 47–53
time model, 52
Environment Management Infrastructure (EMI), 189–196
Ergonomics design, 202
Externalization, 82, 88

Fact
Brute, 40, 191
Institutional, 40, 164, 191
Function
Brute, 191
Epistemic, 86
Institutional, 191
Purposive, 84

Goals, 84
declarative, 183
GOLEM, 51

Hospital
scenario, 153

Influences and Reactions, 44
Internalization, 88
ISLANDER, 21

J-Moise+, 29
Jason, 79

Lifeworld, 50
Link Interface, *see* Artifacts, Linkability
Localities, 52

MadKit, 51
MASQ, 39
Moise, 26–34, 152–163
Deontic specification, 160
Functional specification, 156
Hospital scenario, 229
Organizational Entity, 28–30
Structural specification, 153

NOPL, 163–171
Facts, 165
Norms, 167
Rules, 166
translation rules, 164
Normative artifact, 41
Normative objects/places, 39
Normative Programming Language (NPL), 31
Norms
in Moise, 160
declarative, 183
in NOPL, 167
in NPL, 31
Normative systems, 22–25
violation, 33

Objective (vs Subjective) Coordination, 146
Obligations, *see* Norms
ORA4MAS, 174
Organization Centered MAS (OCMAS), 17
Organization Modeling Language (OML), 18, 152
Organizational Entity (OE), 18
Time, 31

Organizational Management Infrastructure (OMI), 18
based on artifacts, 173–184

Perception
in CArtAgO, 70
Focusing, 70
Observation, 71
model, 48
Permissions, *see* Norms
Producers-Consumers
scenario, 83
Prohibitions, *see* Norms

S-Moise$^+$, 28
Searle, John, 40, 191
Situated Electronic Institutions (SEI), 42
Sociotechnical systems, 202

Task environment, 44
Time, 52
Organizational Entity, 158
Workspace, 124, 133
Workspace vs. Organization, 165

Usage Interface, *see* Artifacts Operations

Work Environment, 55
Workspace, 56
configuration, 102
entering and leaving, 72
Time, 133
Workspace Rules, 131–142, 204–208
Dynamics, 136
Examples, 142
Syntax, 134

Bibliography

[1] P. Agre and I. Horswill. Lifeworld analysis. *Journal of Artificial Intelligence Reserach*, 6:111–145, 1997.

[2] H. Aldewereld. *Autonomy vs. Conformity - an Institutional Perspective on Norms and Protocols*. PhD thesis, Utrecht University, SIKS dissertation series, 2007.

[3] H. Aldewereld, S. lvarez Napagao, F. Dignum, and J. Vazquez-Salceda. Making norms concrete. In *Ninth International Conference on Agents and Multiagent Systems (AAMAS-2010)*, 2010.

[4] G. Andrighetto, M. Campenní, R. Conte, and M. Paolucci. On the immergence of norms: a normative agent architecture. In *Proceedings of AAAI Symposium on Social and Organizational Aspects of Intelligence*, 2007.

[5] I. E. Association. What is ergonomics. `http://iea.cc/`, 2009.

[6] J.-A. Báez-Barranco, T. Stratulat, and J. Ferber. A unified model for physical and social environments. In *Environments for Multi-Agent Systems III, Third International Workshop (E4MAS 2006)*, volume 4389 of *Lecture Notes in Computer Science*, pages 41–50. Springer, 2006.

[7] M. Baldoni, V. Genovese, and L. van der Torre. Adding Organizations and Roles as primitives to the JADE framework. In *Proc. of the 3rd International Workshop on Normative MAS*, 2008.

[8] F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley, 2007.

[9] G. Boella and L. van der Torre. Regulative and constitutive norms in normative multiagent systems. In *International conference on Knowledge and Representation (KR 2004)*, 2004.

[10] O. Boissier and B. Gâteau. Normative multi-agent organizations: Modeling, support and control, draft version. In *Schloss Dagstuhl - Leibniz-Zentrum fr Informatik, Dagstuhl Seminar Proceedings. 07122 - Normative Multi-agent Systems*, 2007.

[11] O. Boissier, J. F. Hübner, and J. S. Sichman. Organization Oriented Programming: From Closed to Open Organizations. In *Engineering Societies for Agent Worlds (ESAW-2006). Extended and Revised version in Lecture Notes in Computer Science LNCS series, Springer*, pages 86–105, 2006.

[12] O. Boissier, J. F. Hübner, and J. S. Sichman. Organization oriented programming: from closed to open organizations. In G. O'Hare, O. Dikenelli, and A. Ricci, editors, *Engineering Societies in the Agents World VII (ESAW 06)*, volume 4457 of *LNCS*, pages 86–105. Springer-Verlag, 2007.

[13] R. H. Bordini, J. F. Hübner, and R. Vieira. Jason and the Golden Fleece of agent-oriented programming. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*, pages 3–37. Springer-Verlag, 2005.

[14] R. H. Bordini, J. F. Hübner, and M. Wooldrige. *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley Series in Agent Technology. John Wiley & Sons, 2007.

[15] E. Bou, M. López-Sánchez, and J. A. Rodríguez-Aguilar. Adaptation of autonomic electronic institutions through norms and institutional agents. In *Engineering Societies for Agent Worlds (ESAW 2006)*, Lecture Notes in Computer Science (LNCS), pages 300–319. Springer, 2006.

[16] M. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, 1987.

[17] S. Bromuri and K. Stathis. Situating Cognitive Agents in GOLEM. In D. Weyns, S. Brueckner, and Y. Demazeau, editors, *Engineering Environment-Mediated Multiagent Systems (EEMMAS'07)*. LNCS Springer, Oct 2007.

[18] J. Campos, M. Lòopez-Sànchez, J. A. Rodrìguez-Aguilar, and M. Esteva. Formalising Situatedness and Adaptation in Electronic Institutions. In *COIN-08, Proc.*, 2008.

[19] J. Campos, M. López-Sánchez, and M. Esteva. Coordination support in Multi-Agent systems. In *Proceedings of the 8th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-09)*, pages 1301–1302, Richland, SC, 2009. IFAAMAS.

[20] CArtAgO. `http://cartago.sourceforge.net`. Project Home Page.

[21] M. Casadei. *Self-Organising Coordination Systems*. PhD thesis, Alma Mater Studiorum - Università di Bologna, 2008.

[22] C. Castelfranchi. Modeling Social Action for AI Agents. *Artificial Intelligence*, 103:157–182, 1998.

[23] C. Castelfranchi. Engineering Social Order. In *Engineering Societies in Agents World*, volume 1972 of *Lecture Notes Computer Science (LNCS)*, pages 1–18. Springer, 2000.

[24] C. Castelfranchi. *Theories and Practice in Interaction Design*, chapter From Conversation to Interaction via Behavioral Communication, pages 157–179. (S. Bagnara and G. C. Smith eds.) Erlbaum, 2006.

[25] C. Castelfranchi, F. Dignum, C. M. Jonker, and J. Treur. Deliberative normative agents: Principles and architecture. In N. R. Jennings and Y. Lespérance, editors, *6th International Workshop Intelligent Agents VI, Agent Theories, Architectures, and Languages (ATAL-99)*, volume 1757 of *Lecture Notes in Computer Science LNCS*, pages 364–378. Springer, 2000.

[26] C. Castelfranchi and F. Paglieri. The role of Beliefs in Goal dynamics: Prolegomena to a constructive theory of intentions. *Synthese*, 155:237–263, 2007.

[27] M. Castoldi. Da A ad A (Teoria delle Catastrofi). Sony BMG, 2007.

[28] P. Ciancarini. Coordination models and languages as software integrators. *ACM Computing Surveys*, 28(2):300–302, 1996.

[29] V. T. da Silva. From the specication to the implementation of norms: an automatic approach to generate rules from norm to govern the behaviour of agents. *Journal of Autonomous Agents and Multi-Agent Systems*, pages 113–155, 2008.

[30] P. D'Altan, J. Meyer, and R. Wieringa. An integrated framework for ought-to-be and ought-to-do constraints. *Artificial Intelligence and Law*, 4:77–111, 1996.

[31] M. Dastani. 2APL: a practical agent programming language. *Autonomous Agent and Multi-Agent Systems*, 16:214–248, 2008.

[32] M. Dastani, V. Dignum, and F. Dignum. Role-assignment in open agent societies. In *International Joint Conference on Agents and Multi Agent Systems (AAMAS 2003)*, pages 489–496, Melbourne, 2003.

[33] M. Dastani, D. Grossi, J.-J. C. Meyer, and N. A. M. Tinnemeier. Normative Multi-Agent Programs and Their Logics. In *Knowledge Representation for Agents and Multi-Agent Systems, First International Workshop, KRAMAS 2008, Sydney, Australia, Revised Selected Papers*, volume 5605 of *Lecture Notes in Computer Science*. Springer, 2008.

[34] M. Dastani, N. Tinnemeier, and J.-J. C. Meyer. A programming language for normative multi-agent systems. In V. Dignum, editor, *Multi-Agent Systems: Semantics and Dynamics of Organizational Models*. IGI-Global, 2009.

[35] D. Dennett. *The Intentional Stance*. MIT Press, 1987.

[36] L. Dennis, B. Farwer, H. R. Bordini, M. Fisher, and M. Wooldridge. A common semantic basis for BDI languages. In *Programming Multi-Agent Systems*, Lecture Notes in Computer Science (LNCS). Springer Berlin / Heidelberg, 2007.

[37] L. A. Dennis, B. Farwer, R. H. Bordini, and M. Fisher. A flexible framework for verifying agent programs. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 1303–1306, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[38] F. Dignum. Abstract norms and electronic institutions. In *Proceedings of Agent-Based Social Systems: Theories and Applications (RASTA-02)*, pages 93–104, 2002.

[39] F. Dignum, V. Dignum, J. Thangarajah, L. Padgham, and M. Winikoff. Open agent systems ??? In *Agent-Oriented Software Engineering VIII*,

volume 4951 of *Lecture Notes Computer Science*, pages 73–87. Springer, 2008.

[40] M. V. F. d. A. J. G. Dignum. *A model for organizational interaction: based on agents, founded in logic*. PhD thesis, Utrecht University, SIKS dissertation series 2004-1, 2003.

[41] V. Dignum, editor. *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*. IGI-Global, 2009.

[42] V. Dignum. The role of organization in agent systems. In V. Dignum, editor, *Multi-Agent Systems: Semantics and Dynamics of Organizational Models*. IGI-Global, 2009.

[43] V. Dignum and F. Dignum. What's in it for me? agent deliberation on taking up social roles. In *European Workshop on Multi-Agent Systems (EUMAS 2004)*, 2004.

[44] F. E. Emery and E. L. Trist. Socio-technical Systems. *Management Sciences Models and Techniques*, 2, 1960.

[45] M. Esteva, D. de la Cruz, and C. Sierra. Islander: an electronic institutions editor. In *First international joint conference on Autonomous agents and Multiagent Systems (AAMAS - 02)*, pages 1045–1052, 2002.

[46] M. Esteva, J. Padget, and C. Sierra. Formalizing a language for institutions and norms. In *Intelligent Agents VIII*, volume 2333 of *LNAI*, pages 348–366. Springer, 2002.

[47] M. Esteva, J. A. Rodríguez-Aguilar, B. Rosell, and J. L. AMELI: An agent-based middleware for electronic institutions. In N. R. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, editors, *Proceedings of International conference on Autonomous Agents and Multi Agent Systems (AAMAS'04)*, pages 236–243, New York, 2004. ACM.

[48] M. Esteva, J. A. Rodríguez-Aguilar, C. Sierra, P. Garcia, and J. L. Arcos. On the Formal Specifications of Electronic Institutions. In *Agent Mediated Electronic Commerce, The European AgentLink Perspective.*, pages 126–147, London, UK, 2001. Springer-Verlag.

[49] J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In Y. Demezeau, editor, *Proceedings of the 3rd International conference on Multi-Agent Systems (ICMAS'98)*, pages 128–135. IEEE Press, 1998.

[50] J. Ferber, O. Gutknecht, and F. Michel. From Agents to Organizations: An Organizational View of Multi-agent Systems. In *Proceedings of (AOSE-03)*, volume 2935 of *Lecture Notes Computer Science (LNCS)*. Springer, 2003.

[51] J. Ferber and J.-P. Müller. Influences and Reaction: a Model of Situated Multi-Agent Systems. In *Proc. of the 2nd Int. Conf. on Multi-Agent Systems (ICMAS'96)*. AAAI, 1996.

[52] FIPA. `http://www.fipa.org`. Foundation for Intelligent Physical Agents.

[53] A. García-Camino, J. A. Rodríguez-Aguilar, C. Sierra, and W. Vasconcelos. Constraining rule-based programming norms for electronic institutions. *Journal of Autonomous Agents and Multi-Agent Systems*, 18(1):186–217, 2009.

[54] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.

[55] D. Gelernter and N. Carriero. Coordination languages and their significance. *Commun. ACM*, 35(2):96, 1992.

[56] D. Grossi, H. Aldewered, and F. Dignum. *Ubi Lex, Ibi Poena*: Designing norm enforcement in e-institutions. In *COIN-07, Proceedings*, vol.4386, LNAI. Springer, 2007.

[57] O. Gutknecht and J. Ferber. The MADKIT agent platform architecture. In *Agents Workshop on Infrastructure for Multi-Agent Systems*, pages 48–55, 2000.

[58] A. Helleboogh, G. Vizzari, A. Uhrmacher, and F. Michel. Modeling dynamic environments in multi-agent simulation. *Autonomous Agents and Multi-Agent Systems*, 14(1):87–116, 2006.

[59] C. Hewitt. Perfect Disruption: The Paradigm Shift from Mental Agents to ORGs. *IEEE Internet Computing*, 13, 2009.

[60] J. F. Hübner, , J. S. Sichman, and O. Boissier. Developing organised multi-agent systems using the MOISE+ model: Programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, 1(3/4):370–395, 2007.

[61] J. F. Hübner, O. Boissier, and R. H. Bordini. Normative programming for organisation management infrastructures. In *MALLOW Workshop on Coordination, Organization, Institutions and Norms in Agent Systems in Online Communities (COIN-MALLOW 2009)*, 2009.

[62] J. F. Hübner, O. Boissier, and R. H. Bordini. From Organisation Specification to Normative Programming in Multi-Agent Organisation Management. In *(submitted to) International Conference on Agents and Multi Agent Systems (AAMAS-10)*, 2010.

[63] J. F. Hübner, O. Boissier, R. Kitio, and A. Ricci. Instrumenting Multi-Agent Organisations with Organisational Artifacts and Agents. *Journal of Autonomous Agents and Multi-Agent Systems*, April 2009.

[64] J. F. Hübner, J. S. Sichman, and O. Boissier. A Model for the Structural, Functional, and Deontic Specification of Organizations in Multiagent Systems. In *SBIA '02: Proceedings of the 16th Brazilian Symposium on Artificial Intelligence*, pages 118–128, London, UK, 2002. Springer-Verlag.

[65] J. F. Hübner, J. S. Sichman, and O. Boissier. $\mathcal{M}\mathsf{oise}^{+}$: Towards a Structural, Functional, and Deontic model for MAS organization. In C. Castelfranchi and W. L. Johnson, editors, *First International Conference on Agents and Multi-Agent Systems (AAMAS'02)*, pages 501–502. ACM Press, 2002.

[66] J. F. Hübner, J. S. Sichman, and O. Boissier. S-moise^{+}: A middleware for developing organised multi-agent systems. In O. Boissier, J. A. Padget, V. Dignum, G. Lindemann, E. T. Matson, S. Ossowski, J. S. Sichman, and J. Vázquez-Salceda, editors, *Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems, AAMAS 2005 International Workshops*, volume 3913 of *Lecture Notes in Computer Science*, pages 64–78. Springer, 2005.

[67] J. F. Hübner, J. S. Sichman, and O. Boissier. Developing Organised Multi-Agent Systems Using the $\mathcal{M}$oise Model: Programming Issues at the System and Agent Levels. *Agent-Oriented Software Engineering*, 1(3/4):370–395, 2007.

[68] Jadex. `http://vsis-www.informatik.uni-hamburg.de/projects/jadex/`. Project Home Page.

[69] Jason. `http://jason.sourceforge.net`. Project Home Page.

[70] N. R. Jennings. An agent-based approach for building complex software systems. *Commun. ACM*, 44(4):35–41, 2001.

[71] A. J. I. Jones and M. Sergot. On the characterization of law and computer systems. In J.-J. Meyer and R. J. Wieringa, editors, *Deontic logic in computer science: Normative System Specification*. John Wiley and Sons, 1993.

[72] T. Juan, A. R. Pearce, and L. Sterling. Roadmap: extending the gaia methodology for complex open systems. In *Proceedings of the first international joint conference on Agents and Multiagent Systems (AAMAS-02)*, pages 3–10. ACM Press, 2002.

[73] A. Kaplan. *The Conduct of Inquiry: Methodology for Behavioral Science*. Chandler Publishing Co., 1964.

[74] J. Kepler. *Astronomia Nova*. , 1609.

[75] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP-97)*, 1997.

[76] D. Kirsh. The intelligent use of space. *Artificial Intelligence*, 73(1-2):31–68, 1995.

[77] D. Kirsh. Distributed cognition, coordination and environment design. In *Proceedings of theEuropean Cognitive Science Society*, 1999.

[78] D. Kirsh and P. Maglio. On distinguishing epistemic from pragmatic action. *Cognitive Science: A Multidisciplinary Journal*, 18(4):513–549, 1994.

[79] B. Kristensen. Object-oriented modeling with Roles. In *Proc. of the 2nd Int. Conf. on Object Oriented Information System*, 1995.

[80] A. N. Leontjev. *Activity, Consciousness, and Personality*. Prentice Hall, 1978.

[81] E. Lorini and M. Piunti. Introducing Relevance Awareness in BDI Agents. In L. Braubach, J.-P. Briot, and J. Thangarajah, editors, *Seventh international Workshop on Programming Multi-Agent Systems (PROMAS-09) - Revised and Extended version*, volume 5919 of *Lecture Notes in Artificial Intelligence*. Springer, 2009.

[82] T. Malone and K. Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119, 1994.

[83] F. R. Meneguzzi and M. Luck. Norm-based behaviour modification in bdi agents. In C. Sierra, C. Castelfranchi, K. S. Decker, and J. S. Sichman, editors, *AAMAS (1)*, pages 177–184. IFAAMAS, 2009.

[84] Moise. `http://moise.sourceforge.net`. Project Home Page.

[85] B. A. Nardi. *Context and Consciousness: Activity Theory and Human-Computer Interaction*. MIT Press, 1996.

[86] I. Newton. *Philosophiae Naturalis Principia Mathematica: the Third edition.* , 1726.

[87] D. Norman. Cognitive Artifacts. In *Designing interaction: Psychology at the human–computer interface*. Cambridge University Press, New York, 1991.

[88] D. A. Norman. *The Design of Everyday Things*. The Mit Press, 1998.

[89] J. Odell, H. V. D. Parunak, M. Fleischer, and S. Brueckner. Modeling agents and their environment. In *Agent-Oriented Software Engineering III, Third International Workshop, AOSE 2002, Bologna, Italy, July 15, 2002, Revised Papers and Invited Contributions*, volume 2585 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2003.

[90] J. J. Odell, H. Van, H. V. D. Parunak, and M. Fleischer. The role of roles in designing effective agent organizations. In *Software Engineering for Large-Scale Multi-Agent Systems, LNCS 2603*, volume 2603 of *LNCS*, pages 27–38. Springer, 2003.

[91] F. Y. Okuyama, R. H. Bordini, and A. C. da Rocha Costa. A distributed normative infrastructure for situated multi-agent organisations. In *Declarative Agent Languages and Technologies VI*, volume 5397 of *Lecture Notes Computer Science (LNCS)*. Springer, 2009.

[92] A. Omicini. Towards a notion of agent coordination context. In D. C. Marinescu and C. Lee, editors, *Process Coordination and Ubiquitous Computing*, chapter 12, pages 187–200. CRC Press, Boca Raton, FL, USA, Oct. 2002.

[93] A. Omicini and E. Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3):277–294, Nov. 2001.

[94] A. Omicini and E. Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3):277294, 2001.

[95] A. Omicini and S. Ossowski. Objective versus subjective coordination in the engineering of agent systems. In M. Klusch, S. Bergamaschi, P. Edwards, and P. Petta, editors, *Intelligent Information Agents: An AgentLink Perspective*, volume 2586 of *LNAI: State-of-the-Art Survey*, pages 179–202. Springer-Verlag, Mar. 2003.

[96] A. Omicini, M. Piunti, A. Ricci, and M. Viroli. Agents, intelligence, and tools. In M. Bramer, editor, *Artificial Intelligence: An International Perspective*, volume 5640 of *LNAI: State-of-the-Art Survey*, chapter 9, pages 157–173. Springer, 2009.

[97] A. Omicini, A. Ricci, and M. Viroli. *Agens Faber*: Toward a theory of artefacts for MAS. *Electronic Notes in Theoretical Computer Sciences*, 150(3):21–36, 29 May 2006. 1st International Workshop "Coordination and Organization" (CoOrg 2005), COORDINATION 2005, Namur, Belgium, 22 Apr. 2005. Proceedings.

[98] A. Omicini, A. Ricci, and M. Viroli. Coordination artifacts as first-class abstractions for MAS engineering: state of the research. *Software Engineer-*

ing for Multi-Agent Systems IV: Research Issues and Practical Application, Invited Paper, volume 3914 of LNAI:7190, 2006.

[99] A. Omicini, A. Ricci, and M. Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17 (3), 2008.

[100] A. Omicini, A. Ricci, M. Viroli, C. Castelfranchi, and L. Tummolini. Coordination Artifacts: Environment-based Coordination for Intelligent Agents. In *Proceedings of Joint Conference Autonomous Agents and Multi Agent Systems (AAMAS-04)*, volume 1, pages 286–293, New York, USA, 2004.

[101] A. Omicini and F. Zambonelli. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, Sept. 1999.

[102] H. V. D. Parunak, S. Brueckner, and J. A. Sauter. Digital pheromone mechanisms for coordination of unmanned vehicles. In *The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy, Proceedings*, pages 449–450. ACM, 2002.

[103] W. A. Pasmore. *Designing Effective Organizations : the Sociotechnical Systems perspective*. Wiley series on organizational assessment and change. Wiley, 1988.

[104] S. Pheasant and C. M. Haslegrave. *Bodyspace: Anthropometry, Ergonomics and the Design of Work*. Taylor & Francis, Inc., 2005.

[105] M. Piunti and A. Ricci. From Agents to Artifacts Back and Forth: Purposive and Doxastic use of Artifacts in MAS. In *Proceedings of Sixth European Workshop on Multi-Agent Systems(EUMAS-08)*, Bath, UK., 2008.

[106] M. Piunti and A. Ricci. Cognitive Use of Artifacts: Exploiting Relevant Information Residing in MAS Environments. In J.-J. Meyer and J. Broersen, editors, *Knowledge Representation for Agents and Multi-Agent Systems*, volume 5605 of *LNAI*, pages 114–129. Springer, 2009. 1st International Workshop (KRAMAS 2008), Sydney, Australia, 17 Sept. 2008, Revised Selected Papers.

[107] M. Piunti, A. Ricci, O. Boissier, and J. F. Hübner. Embodied Organisations in MAS Environments. In L. Braubach, W. van der Hoek, P. Petta, and A. Pokahr, editors, *MATES*, volume 5774 of *Lecture Notes in Computer Science*, pages 115–127. Springer, 2009.

[108] M. Piunti, A. Ricci, O. Boissier, and J. F. Hübner. Embodying Organisations in Multi-agent Work Environments. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT 2009)*, pages 511–518, Milan, Italy, 2009. IEEE.

[109] M. Piunti, A. Ricci, L. Braubach, and A. Pokahr. Goal-directed Interactions in Artifact-Based MAS: *Jadex* Agents playing in CArtAgO Environments. In *IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT 2008)*, Sydney, NSW, Australia., 2008.

[110] M. Piunti, A. Ricci, and A. Santi. Soa/ws applications using cognitive agents working in cartago environments. In *Proceedings of 10th Joint Conference AI*IA TABOO From Objects to Agents (WOA 2009)*, 2009.

[111] M. Piunti, A. Santi, and A. Ricci. Programming SOA/WS Systems with BDI Agents and Artifact-Based Environments. In *In Proceedings of MALLOW federated Workshops on Agents, Web Services and Ontologies, Integrated Methodologies (AWESOME-09)*, Torino, September 2009.

[112] A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: A BDI Reasoning Engine. In R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, editors, *Multi-Agent Programming*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 149–174. Springer, 2005.

[113] N. Prasad, K. Decker, and A. Garvey. Exploring organizational designs with taems: A case study of distributed data processing. In *2nd International conference on Multi Agent Systems (ICMAS 96)*, 1996.

[114] A. S. Rao and M. P. Georgeff. BDI Agents: From Theory to Practice. In *First International Conference on Multi Agent Systems (ICMAS95)*, 1995.

[115] A. Ricci, A. Omicini, and E. Denti. Activity Theory as a framework for MAS coordination. In P. Petta, R. Tolksdorf, and F. Zambonelli, editors,

Engineering Societies in the Agents World III, volume 2577 of *LNCS*, pages 96–110. Springer-Verlag, Apr. 2003. 3rd International Workshop (ESAW 2002), Madrid, Spain, 16–17 Sept. 2002. Revised Papers.

[116] A. Ricci and M. Piunti. Implementing over-sensing in heterogeneous multi-agent systems on top of artifact-based environments. In L. Braubach, W. van der Hoek, P. Petta, and A. Pokahr, editors, *Proceedings of Seventh German conference on Multi-Agent System Technologies (MATES 2009)*, volume 5774 of *Lecture Notes in Computer Science*, pages 232–237. Springer, 2009.

[117] A. Ricci, M. Piunti, L. D. Acay, R. Bordini, J. Hubner, and M. Dastani. Integrating artifact-based environments with heterogeneous agent-programming platforms. In *Proceedings of 7th International Conference on Agents and Multi Agents Systems (AAMAS08)*, 2008.

[118] A. Ricci, M. Piunti, and M. Viroli. Externalisation and Internalization: A New Perspective on Agent Modularisation in Multi-Agent Systems Programming. In M. Dastani, A. E. F. Seghrouchni, J. Leite, and P. Torroni, editors, *Proceedings of MALLOW 2009 federated workshops: LAnguages, methodologies and Development tools for multi-agent systemS (LADS 2009)*, September 2009.

[119] A. Ricci, M. Piunti, M. Viroli, and A. Omicini. Environment programming in CArtAgO. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah-Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications, Vol. 2*, pages 259–288. Springer, 2009.

[120] A. Ricci, A. Santi, and M. Piunti. Action and Perception in Multi-Agent Programming Languages: From Exogenous to Endogenous Environments. In *Proceedings of workshop Programming Multiagent Systems (PROMAS-10)*, 2010.

[121] A. Ricci, M. Viroli, and A. Omicini. The A&A programming model & technology for developing agent environments in MAS. In M. Dastani, A. El Fallah Seghrouchni, A. Ricci, and M. Winikoff, editors, *Programming Multi-Agent Systems*, volume 4908 of *LNAI*, pages 91–109. Springer, 2007.

[122] A. Ricci, M. Viroli, and A. Omicini. CArtAgO: A framework for prototyping artifact-based environments in MAS. In D. Weyns, H. V. D. Parunak, and F. Michel, editors, *Environments for MultiAgent Systems III*, volume 4389 of *LNAI*, pages 67–86. Springer, Feb. 2007.

[123] A. Ricci, M. Viroli, and G. Piancastelli. simpA: A simple agent-oriented Java extension for developing concurrent applications. In M. Dastani, A. E. F. Seghrouchni, J. Leite, and P. Torroni, editors, *Languages, Methodologies and Development Tools for Multi-Agent Systems (LADS 2007)*, volume 5118 of *LNAI*, pages 176–191. Springer-Verlag: Heidelberg, Germany, Durham, UK, 2007.

[124] A. Ricci, M. Viroli, and M. Piunti. Formalising the Environment in MAS Programming: A Formal Model for Artifact-Based Environments. In L. Braubach, J.-P. Briot, and J. Thangarajah, editors, *Proceedings of AAMAS Workshop Programming Multi-Agent Systems (PROMAS-09)*, Lecture Notes in Artificial Intelligence. Springer, 2009.

[125] S. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach (2nd ed.)*. Prentice Hall, 2003.

[126] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role base access control models. *Computer*, 29(2):38–47, 1996.

[127] J. R. Searle. *Speech Acts*, chapter What is a Speech Act? Cambridge University Press, 1964.

[128] J. R. Searle. *The Construction of Social Reality*. Free Press, 1997.

[129] S. Sen and S. Airiau. Emergence of norms through social learning. In *Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 2007.

[130] J. M. Serrano and S. Saugar. Operational semantics of Multiagent Interactions. In *Proceedings of the 6th international joint conference on Autonomous Agents and Multiagent Systems (AAMAS-07)*, pages 1–8, New York, NY, USA, 2007. ACM.

[131] Y. Shoham and M. Tennenholtz. On social laws for artificial agent societies: off-line design. *Artificial Intelligence*, 73(1-2):231–252, 1995.

[132] T. Stratulat, J. Ferber, and J. Tranier. MASQ: Towards an Integral Approach of Agent-Based Interaction. In *Proc. of 8th Conf. on Agents and Multi Agent Systems (AAMAS-09)*, 2009.

[133] T. Susi and T. Ziemke. Social cognition, artefacts, and stigmergy: A comparative analysis of theoretical frameworks for the understanding of artefact-mediated collaborative activity. *Cognitive Systems Research*, 2(4):273–290, Dec. 2001.

[134] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[135] N. Tinnemeier, M. Dastani, J.-J. Meyer, and L. van der Torre. Programming normative artifacts with declarative obligations and prohibitions. In *IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT 2009)*, 2009.

[136] M. B. van Riemsdijk, M. Dastani, and M. Winikoff. Goals in agent systems: A unifying framework. In *Proceedings of the seventh international joint conference on Autonomous agents and multiagent systems (AAMAS 08)*, 2008.

[137] M. B. van Riemsdijk, K. Hindriks, and C. Jonker. Programming organisation-aware agents: a research agenda. In *In 10th Engineering Societies for Agents Worlds (ESAW 09)*, 2009.

[138] J. Vázquez-Salceda, V. Dignum, and F. Dignum. Organizing multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 11(3):307–360, 2005.

[139] R. Vieira, A. Moreira, M. Wooldridge, and R. H. Bordini. On the formal semantics of speech-act based communication in an agent-oriented programming language. *Journal of Artificial Intelligence Ressearch*, 29(1):221–267, 2007.

[140] M. Viroli, A. Ricci, and A. Omicini. Operating instructions for intelligent agent coordination. *The Knowledge Engineering Review*, 21(1):49–69, Mar. 2006.

[141] P. Wegner and D. Goldin. Computation beyond Turing machines. *Communication of ACM*, 46(4):100–102, 2003.

[142] D. Weyns and T. Holvoet. Formal model for situated multiagent systems. *Fundamenta Informaticae*, 63(2–3):125–158, 2004.

[143] D. Weyns and T. Holvoet. A reference architecture for situated multiagent systems. In *Environments for Multiagent Systems III*, volume 4389 of *Lecture Notes in Computer Science*, pages 1–40. Future University, Hakodate, Japan, Springer, 2007.

[144] D. Weyns, A. Omicini, and J. J. Odell. Environment as a first-class abstraction in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, Feb. 2007. Special Issue on Environments for Multi-agent Systems.

[145] D. Weyns and H. V. D. Parunak, editors. *Journal of Autonomous Agents and Multi-Agent Systems. Special Issue: Environment for Multi-Agent Systems*, volume 14 (1). Springer Netherlands, 2007.

[146] D. Weyns, H. V. D. Parunak, F. Michel, T. Holvoet, and J. Ferber. Environments for multiagent systems: State-of-the-art and research challenges. In D. Weyns, H. V. D. Parunak, F. Michel, T. Holvoet, and J. Ferber, editors, *Environment for Multi-Agent Systems*, volume 3374, pages 1–47. Springer-Verlag, Berlin-Heidelberg, 2005.

[147] D. Weyns, E. Steegmans, and T. Holvoet. Towards active perception in situated multiagent systems. *Applied Artificial Intelligence*, 18(9–10):867–883, 2004.

[148] M. Wooldridge. *An Introduction to Multi-Agent Systems*. John Wiley & Sons, Ltd, 2002.

[149] WSIT. Web services interoperability technologies. `http://wsit.dev.java.net`.

[150] F. Zambonelli, N. R. Jennings, and M. Wooldridge. Organisational abstractions for the analysis and design of multi-agent systems. In *Agent-Oriented Software Engineering: First International Workshop, AOSE 2000 Limerick, Ireland, June 10, 2000 Revised Papers*, pages 235–251, Berlin, Heidelberg, 2001. Springer-Verlag.