ÉCOLE DOCTORALE SIS

SCIENCES, INGÉNIERIE, SANTÉ

N° d'ordre : XXXXX

MÉMOIRE DE THÈSE

pour obtenir le titre de

Docteur en Sciences

de l'École des Mines - Saint-Étienne

Spécialité : Informatique

Présentée et soutenue par

Camille Persson

A Decentralized and Distributed Adaptive Governance for Machine-to-Machine Systems

A Multi-Agent Oriented Programming Approach

Thèse préparée à l'Institut Fayol
Dans le cadre du projet FUI-Minalogic SensCity
Soutenue le 31 Octobre 2014

Jury :

Président :	Prénom Nom	*Titre*	Affiliation
Rapporteurs :	Michel Occello	*Professeur*	UPMF - Valence
	Laurent Vercouter	*Professeur*	INSA - Rouen
Directeur :	Olivier Boissier	*Professeur*	ENS Mines - Saint-Étienne
Encadrants :	Gauthier Picard	*Maître Assistant*	ENS Mines - Saint-Étienne
	Fano Ramparany	*Ingénieur R&D*	Orange Labs - Grenoble
Examinateur :	Yves Demazeau	*Directeur de Recherche*	CNRS - Grenoble

Acknowledgment

Abstract

In this thesis dissertation, we address the problem of Machine-to-Machine (M2M) systems governance.

M2M systems are based on sensors and actuators networks immersed in the physical world. They differ from classical information technology systems in many aspects. First, M2M devices are very constrained in terms of computing power, network and energy supply.

Then, so-called “vertical” approaches (one stakeholder is involved from end-to-end) are very expensive. Thus, a growing trend emerges for so-called “horizontal” architectures, where several stakeholders collaborate within the same infrastructure.

Finally, the growing number of connected devices raises the problem of scalability management.

We propose an M2M governance based on a Multi-Agent Oriented Programming (MAOP) approach, divided into the different dimensions of the Multi-Agent System (MAS).

The organization expresses the governance strategy through an horizontal and vertical specifications, corresponding to each stakeholder’s requirements.

Agent expresses the governance tactic balancing the horizontal and vertical requirements. They reason about the governance specification in order to make it evolve with respect to the M2M system’s constraints and to handle scalability issues. In addition, they decide which policies to use for managing infrastructure adaptations.

Finally, artifacts provide an abstraction for the governed system to the agents, and primitives for managing governance policies and activate them.

We propose an implementation of our approach called AGaMeMon (Adaptive Governance MAS for M2M systems), build with the JaCaMo framework.

We apply our model to the SensCity project, which involves several stakeholders, and which was a starting point for this thesis at Orange Labs (CIFRE). Finally, we validate and discuss our approach experimentally by the means of simulations.

Nous appliquons notre modèle au projet SensCity, impliquant plusieurs acteurs, qui a servi de base aux travaux de cette thèse CIFRE chez Orange Labs. Nous validons et discutons notre approche de manière expérimentale à l’aide de simulations.

Résumé

Dans cette thèse, nous posons le problème de la gouvernance des systèmes M2M.

Les systèmes M2M sont basés sur des réseaux de capteurs et d'effecteurs immergés dans le monde physique. Ils se distinguent des systèmes d'information classiques par plusieurs aspects. D'abord, les appareils sont fortement contraints en terme de puissance de calcul, de réseau et d'autonomie d'énergie.

Ensuite, les approches dites "verticales" (une entreprise intervient de bout-en-bout) sont extrêmement chères. Ainsi, une tendence émerge pour des architectures dites "horizontales" où plusieurs parties-prenantes collaborent au sein de la même infrastructure.

Enfin, le nombre grandissant d'objets connectés pose le problème du passage à l'échelle.

Nous proposons une gouvernance de systèmes M2M basée sur une approche par Programmation Orientée Multi-Agent (MAOP), décomposée selon les différentes dimensions du Système Multi-Agent (SMA).

L'organisation exprime la stratégie de gouvernance à travers des spécifications horizontale et verticales, exprimant les besoins de toutes les parties-prenantes.

Les agents expriment la tactique de gouvernance conciliant les besoins horizontaux et verticaux. Ils raisonnent sur les spécifications de la stratégie afin de la faire évoluer et répondre aux contraintes du système et gérer le passage à l'échelle. De plus, ils décident de la politique à utiliser pour gérer les adaptations de l'infrastructure.

Enfin, des artefacts fournissent une couche d'abstraction du système gouverné aux agents, ainsi que les primitive nécessaires pour gérer les politiques d'adaptation et les appliquer.

Nous proposons une implémentation de cette approche nommée AGaMeMon (Adaptive Governance MAS for M2M systems – SMA de Gouvernance Adaptative pour les systèmes M2M) à l'aide du framework JaCaMo.

Nous appliquons notre modèle au projet SensCity, impliquant plusieurs acteurs, qui a servi de base aux travaux de cette thèse CIFRE chez Orange Labs. Enfin, nous validons et discutons notre approche de manière expérimentale à l'aide de simulations.

Contents

List of Figures

List of Tables

List of Listings

List of Acronyms

6LoWPAN	IPv6 Low power Wireless Personal Area Network
A&A	Agents & Artifacts
ACL	Agent Communication Language
AOSE	Agent Oriented Software Engineering
AUML	Agent Unified Modelisation Language (Agent UML)
BDI	Belief–Desire–Intention Agent Model
CBSE	Component-Based Software Engineering
CobiT	Control Objectives for Information and related Technology
CoAP	Constrained Application Protocol
CoRE	IETF Constrained RESTful Environments Working Group
DSCL	Network Service Capability Layer (ETSI)
ECA	Event-Condition-Action
EJB	Entreprise Java Bean (CBSE support in Java)
ETSI TC M2M	European Telecommunications Standards Institute Technical Committee on M2M
FIPA	Foundation for Intelligent Physical Agents (IEEE Computer Society standards organization)
FIPA-ACL	FIPA Agent Communication Language
GORE	Goal Oriented Requirement Engineering
GSCL	Gateway Service Capability Layer (ETSI)
IEEE	Institute of Electrical and Electronical Engineers
IEEE WPAN TG4	IEEE Wireless Personal Area Network Task Group 4
IETF	Internet Engineering Task Force
ISACA	Information Systems Audit and Control Association
IT	Information Technology (a.k.a. Information and Communication Technology (ICT))
ITGI	IT Governance Institute

IoT	Internet of Things
KSE	Knowledge Sharing Effort (Advanced Research Project Agency initiative)
KQML	Knowledge Query and Manipulation Language
LLN	Low Power and Lossy Network
M2M	Machine-to-Machine
MAOP	Multi-Agent Oriented Programming
MAPE-K	Monitor–Analyze–Plan–Execute–Knowledge (Autonomic control loop)
MAS	Multi-Agent System
NSCL	Network Service Capability Layer (ETSI)
NFR	Non Functional Requirement
OE	Organizational Entity (MAS)
OS	Organizational Specification (MAS)
QoS	Quality of Service
REST	REpresentational State Transfer
SCL	Service Capability Layer
SLA	Service Level Agreement
SLO	Service Level Objective
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SoS	System of Systems
Telco	Telecommunication Operator
UDDI	Universal Description Discovery and Integration
ULS	Ultra Large Scale Systems
URI	Unique Resource Identifier
WS	Web Service
WSAN	Wireless Sensor and Actuator Network (a.k.a. Wireless Sensor Network (WSN))
WSDL	Web Services Description Language
WSLA	Web Services Level Agreement

INTRODUCTION

This thesis dissertation addresses the problem of governing Machine-to-Machine (M2M) infrastructures. The aim is to enhance *horizontal* M2M infrastructure deployments by enabling several stakeholders to share the deployment and maintenance efforts and cost, instead of pure *vertical* deployments.

To do so, we propose an adaptive governance that takes into account each stakeholder's constraints and balances them for ensuring the overall system's requirements. Our approach is based on the use of a Multi-Agent System (MAS) to support and automate the different governance layers.

In this introduction, we first present the context and challenges of our work, before introducing our contribution and the thesis defended by this document. Finally, we give an overview of the remainder of the manuscript.

Context

M2M refers to networks connecting together web application services and pervasive sensors and actuator –i.e. devices embedded in the real world. Due to the devices' cheap cost, M2M is a key technology for enabling the Internet of Things (IoT), especially for city scale urban deployments.

A typical M2M infrastructure is based on an N-tier architecture model, which can be divided into three domains: application domain, network domain and device domain. The application domain is composed of pervasive applications, also called client applications, providing services to the end-users. The network domain is composed of an M2M platform, enabling the applications to access to sensors and actuators, broadband network infrastructures. The device domain is composed of Wireless Sensor and Actuator Network (WSAN), which is connected to the Internet by an M2M gateway.

Such a WSAN typically use a Low Power and Lossy Network (LLN) technology, enabling the devices' battery life to last longer. In fact, while wireless sensors and actuators are cheap, deployment and human intervention costs are expensive. As a consequence, there is a growing trend for mutualization in order to share the infrastructure deployment and maintenance cost.

Two types of M2M infrastructures can be identified: vertical and horizontal infrastructures.

A *vertical M2M infrastructure* is fully managed by a single stakeholder –or closed collaboration between predefined stakeholders– which assumes itself WSAN deployment, network connection and end-user services. Such a vertical infrastructure guarantee a full control of M2M resources usage and constraints management.

Nevertheless, it implies an high financial effort, which can quickly become unaffordable for large scale applications. It also requires such companies to be qualified in all three M2M domains. In addition, reuse and cross-use of M2M devices is hardly supported, and so, it limits the number of services provided to end-users.

An *horizontal M2M infrastructure* is mutualized between different stakeholders, each of them being specialized in its own domain. Thus, such an horizontal infrastructure would also improves the infrastructure quality. Further more, it eases the openness to additional stakeholders, and so enable large scale M2M deployments and multiplicity of application services, with reasonable costs for each stakeholder.

Nevertheless, beyond the financial and quality benefits, an horizontal M2M deployment raises important issues regarding the constrained nature of M2M resources. In fact, WSAN traffic and M2M devices lifetime are critical in order to maintain an acceptable Quality of Service (QoS) level and maintenance costs.

Challenges and Objectives

Thus, horizontal M2M infrastructures seems to be a promising paradigm that would allow significant advances in the realization of the IoT. Though, such an architecture raises the problem of a governance process adapted to M2M specific constraints.

In fact, beside standardization efforts, such as the European Telecommunications Standards Institute Technical Committee on M2M (ETSI TC M2M), facilitating interoperability between M2M stakeholders, we can identify two main challenges remaining regarding horizontal M2M concerns.

Heterogeneity of Application Requirements. As many stakeholders will share the M2M infrastructure, several client applications may express different requirements in terms of resources to use and QoS. In addition, each end-to-end use of M2M resources should be dynamically defined and able to evolve. We can consider such an end-to-end set of M2M resources as a *virtual vertical deployment*.

Device Lifetime Management. Given the constrained nature of M2M devices and WSAN, and being used by different client applications, with different QoS requirements, device lifetime can be hardly controlled. This is of major importance as human intervention is expensive –e.g. to replace batteries.

Such challenges can be considered as *scalability management issues*. In fact, an horizontal M2M infrastructure enables to deploy a large number of M2M devices providing data to a wide variety of applications. Therefore, an M2M governance should take into account such specific issues in order support horizontal M2M infrastructures.

Governance is a process to control resources usage and ensure that they sustain the enterprise's objectives. That is, a decision cycle that (i) defines resources usage according to business goals, (ii) measures such goals achievement, (iii) analyzes measured performances and (iv) refines business objectives in order to improve such performances and discover new opportunities. Nowadays, governance has been established by Information Technology (IT) management board, for many years, and as shown significant results regarding IT resources efficiency, security, and financial performance.

Since then, SOA governance have been defined to take into account the openness and dynamics introduced by Service Oriented Architecture (SOA). Further more, a recent study revealed that companies' technological choice is mainly driven by governance tools support for such technology –i.e. SOAP is still preferred while REST offers more solutions.

Our objective is to provide an automated governance framework for horizontal M2M infrastructures. Such an objective leads us to define two sub-objectives.

First, we need to define an M2M governance model. That is, a governance model defining how M2M stakeholders can coordinate their business strategies to be integrated into the same infrastructure and information processes.

Based on IT and SOA governance principles, we can identify four levels of M2M governance, we need to address. *Governance strategy* should specify the global M2M objectives and requirements. That is, it should define the objectives of the M2M infrastructure and how every stakeholder should participate in their realization. *Governance tactic* should define intermediate goals and processes in order to fulfill the strategy objectives. To do so, it should provide means to define intermediate goals and decide which governance processes would handle them. *Governance policy* should define means to achieve such intermediate goals by affecting and configuring the M2M resources. Hence, it should constantly monitor the M2M infrastructure's resources in order to control and adapt each its global functioning.

Governance refinement consists of an evaluation cycle of the governance objectives satisfaction to redefine the strategy, improve M2M infrastructure performances, enable financial opportunities and avoid unachievable objectives. That means the governance strategy, tactic and policy should be constantly refined through a cycling process considering the M2M infrastructure performance and evolution.

In addition to such a governance model, we need to define a technological support for embedding the M2M governance process. In fact, given (i) the constrained nature of M2M resources, (ii) the heterogeneity of requirements brought by (iii) the large scale openness of horizontal M2M infrastructures, M2M governance should provide an efficient response the infrastructure's dynamics.

Approach Defended

Given the distributed, heterogeneous, constrained and large scale nature of M2M infrastructure, *we make the hypothesis Multi-Agent Oriented Programming approach provides the adequate levels of abstraction for an efficient M2M governance support*, where each MAS dimension supports a governance level. Following, we describe our different governance levels and how they can be implemented using a MAS.

Our M2M governance approach can be represented by the following five layers: (i) the *business layer* composed of the M2M stakeholders (i.e. companies, management board) involved in the M2M infrastructure, (ii) the *strategy layer* specifying high level objectives and contracts, (iii) the *tactic layer* specifying intermediate goals to fulfill the strategy, (iv) the *policy layer* specifying means for achieving intermediate goals, and (v) the *governed layer* representing the governed M2M infrastructure and legacy components. The scope of our thesis includes the three intermediate layers between the stakeholders and the M2M infrastructure (layers (ii), (iii) and (iv)), as represented in Figure 1. In addition, the *governance refinement* redefines objectives across these three layers.

Following is a description of the different levels of governance abstraction we defined. For each abstraction level, we define its MAS support along the Multi-Agent Oriented Programming (MAOP) dimensions –i.e. *Agent*, *Environment*, *Interaction*, *Organization* dimensions.

Governance Strategy is divided it into two dimensions, namely, the *horizontal* and the *vertical* strategy dimensions.

Horizontal strategy is driven by the ETSI TC M2M standard, which defines *capabilities* and *functionalities* the M2M entities should provide to be interoperable. Vertical strategy specifies requirements for virtual vertical contracts between M2M stakeholders, by the means of Service Level Agreement (SLA) specification.

We choose to support the governance strategy by the MAS *organization*, as it allows to control collective behaviors.

On the one hand, the horizontal organization is specified using roles to represent M2M capabilities and social goals for each M2M functionality. On the other hand, we define a vertical organization template to represent each vertical SLA: roles represents M2M entities involved in such a vertical contract, while terms of the SLA are represented by a social scheme (i.e. a decomposition of social goals grouped into missions).

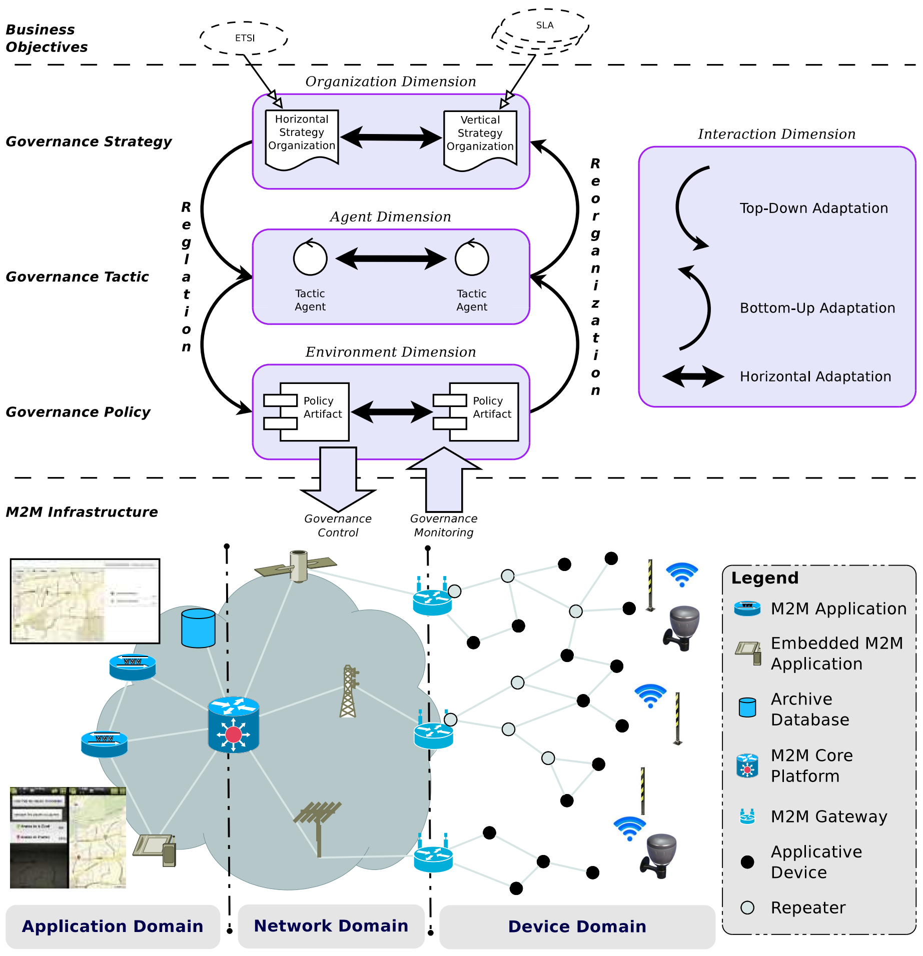


Figure 1: MAOP approach for M2M Governance.

Governance Tactic specifies *governance plans* to decompose strategy objectives into intermediate goals according to the M2M infrastructure state. Such a governance plan is defined by a set of decisions and actions to set up, depending on different contexts.

Governance tactic is supported by the *agent* dimension of the MAS. In fact, agents embody a Belief–Desire–Intention (BDI) decision process that enables high level reasoning capabilities.

Hence, governance plans are implemented as agent plans which evaluate the strategy objectives feasibility and analyze the M2M infrastructure state to adapt it accordingly. We define three different types of agents representing the three types of M2M stakeholders: (i) the *application* provider, (ii) the network *infrastructure* operator, and (iii) the M2M *device* provider.

Governance Policy is defined by a *set of rules specifying the M2M resources usage* in order to achieve the intermediate tactic goals. That is, we define dedicated processes for configuring, monitoring and regulating M2M resources.

We use the MAS *environment* dimension to support such a governance policy, because it provides an abstraction layer for resources to monitor and manipulate them by the means of artifact.

We encapsulate the M2M infrastructure by governance artifacts. Hence, an artifact provides an abstract representation of M2M resources. Such an artifact embeds monitoring functions, and provides configuration and regulation operations allowing the agents to evaluate the environment state and decide of the appropriate policies to set up.

As for agents, we define three types of artifacts, each of them providing specific policies related to the type of M2M entity governed.

Governance Refinement is an evaluation process of the governance performance, performed at each governance level. According to such performance, it *adapts the governance* in order to improve its accuracy. We define three types of refinement: (i) *regulation* resulting from a top-down governance adaptation, triggering new lower-level governance adaptation to enforce upper-level objectives, (ii) *reorganization* which is a bottom-up governance adaptation, resulting from inadequacy of an upper-level objective with respect to lower-level constraints, and (iii) *horizontal* refinement which consists of adapting a governance level to improve governance processes with respect to the M2M infrastructure's evolution.

Governance refinement is supported by the *interaction* dimension of the MAS. Interactions unable communication between the different parts of the MAS inside a dimension and across the other dimensions.

We implement the governance refinement through several adaptation processes. *Regulation* is implemented by top-down adaptation interactions, that is, organization-agent interactions, agent-artifact interactions and artifact-M2M resource interactions. *Reorganization* is implemented by bottom-up adaptation interactions, namely, M2M resource-artifact interactions, artifact-agent interactions, agent-organization interaction. Finally, *horizontal refinement* is performed by adaptation interactions inside a MAS dimension.

Let us note that each of such refinement interactions are independent from each other across the governance levels. That is, refinements are not necessarily propagated from a level to another, which means refinement at a given level may trigger an other type refinement for an other level, or no refinement at all.

Thesis Organization

This introduction provided a brief description of our work. The remainder of this manuscript details our study, motivates our approach and evaluates it. This dissertation is structured into three parts, organized as follows.

Part I State of the Art. First part is a literature survey which aims at stating governance principles to handle M2M issues, and available technologies for supporting such a governance.

In Chapter 1, we review IT and SOA governance to define principles for M2M governance. To do so, we analyze characteristics of an M2M infrastructure and how literature contributes to M2M governance. We end this chapter by a focus on M2M scalability issues and how governance should be defined to tackle such an issue.

Then, we study how M2M governance can be automated thanks to adaptation and control techniques, in Chapter 2. We split our review into two categories: (i) endogenous and (ii) exogenous adaptation and control systems. The goal of this chapter is to define which degree of complexity is necessary to carry an M2M governance system and which approach seems to be the best for it.

Finally, we provide an overview of MAS in Chapter 3 from a governance perspective. We start by reviewing the MAOP literature and then, we analyze how MAS can control and adapt a system, and so carry a full governance cycle.

We conclude the state of the art part by Chapter 4, which provides a synthesis of our literature survey and draws sub-objectives for our proposal.

Part II Proposal for an M2M Governance. In this part, we introduce our proposal for governing M2M infrastructures, supported by an MAOP approach.

We introduce our proposal in Chapter 5. This chapter gives a brief overview of our approach before we detail each layer in the three next chapters.

Chapter 6 presents our organization-based governance strategy. We start by explaining our choice for splitting the strategy into multiple organizations along the horizontal and vertical dimensions of the M2M infrastructure. Then, we describe the horizontal organization, based on the ETSI TC M2M specifications, and a template for vertical organizations. Finally, we explain how the two dimensions are bound together.

In Chapter 7, we describe the agent layer of our governance framework. It is used to implement the governance tactic. First, we describe a general architecture for such governance agents. Then, we extend such design to three types of agents, each of representing a specific M2M point of view.

Finally, we describe our policy layer in Chapter 8. We start by a description of such an artifact design to embed the governance policy and its interfaces. Then, we describe three specific artifacts designed for the different agents' concerns.

Part III Evaluation. Last part evaluates our approach and discusses our contribution.

We describe how we can implement our governance system using the JaCaMo framework, and integrate it into an existing M2M platform, used for the SensCity project, in Chapter 9.

Then, an empirical evaluation of our approach is conducted in Chapter 10. We describe our simulation protocol and analyze the experimental results to show the benefits and drawbacks of our approach.

Finally, in the Publications chapter, we summarize the contribution brought by this thesis. Then, we analyze and discuss its limits in order to draw further work we would proceed.

Part I

State of the Art

Chapter 1

Governance for Machine-to-Machine systems

Contents

1.1 Information Technology (IT) Governance

Historically, the term *governance* in computer science comes from the IT domain where companies where facing two problems (Guldentops et al., 2003): (i) a growing trend for automating corporate processes when (ii) executives could not understand how the IT could add value and increase performance.

In this section, we study the best practices, the principles and the processes used for defining the IT governance. The main contribution in IT governance comes from the Information Systems Audit and Control Association (ISACA)[1] and its affiliated organisation the IT Governance Institute (ITGI)[2] which created the CobiT[3] framework (Hayes et al., 2007). This framework provides processes to the system administrator to define rules for the users and to set up programs to apply these rules (e.g. "white lists" and "black lists" of websites).

Then, the next generation of IT systems brought new issues to the management of such systems. These systems are based on an open architecture, the Service Oriented Architecture (SOA), which facilitates the deployment of new services and their evolution. The openness of such an architecture allows to use and compose services from external stakeholders. Therefore, the governance definition had to evolve. In particular, the service composition becomes an important aspect of governance. This raises new issues for the governance such as contract definition.

This section draws the governance principles for IT systems in Section 1.1.1 and their evolution to manage SOA in Section 1.1.2. This will help us to identify the challenges of governance with respect to the Machine-to-Machine (M2M) issues in Section 1.2.

1.1.1 What is IT Governance ?

Nowadays, IT Governance principles are well established and widely spread. In this section, we focus on the reasons which lead to the need of Governance and the main principles of IT Governance. This study is mainly based on the following reports: Guldentops et al. (2003) which summarizes the ITGI's recommendations, Hayes et al. (2007) which describes the CobiT framework, Toomey (2009) which draws a comprehensive overview of the IT Governance principles, outcomes and issues. In order to improve readability, we will not repeat these references in this section.

1.1.1.1 Motivation: the "elephant in the room"

Since its beginnings, the importance of IT has grown significantly within every companies. If its goal was, at first, to automate resources management such as employees' salary or stock management, IT has become a source of value-added products. In fact, it gives a competitive advantage and increases productivity. Further more, information is now key product to be shared with other stakeholders.

But, while IT's importance grows, its complexity does too. For example, outsourcing leads to more flexibility and increased performance but also less control. So that, it has become the "elephant in the room": the lack of comprehensiveness means a potential emergence of risks. Thus, it is hardly possible to find the source(s) of the problem(s) and to fix it (them) using ad-hoc processes.

[1]Information Systems Audit and Control Association (ISACA): `http://www.isaca.org/`

[2]IT Governance Institute (ITGI): `http://www.itgi.org/`

[3]Control Objectives for Information and related Technology (CobiT): `http://www.isaca.org/COBIT/`

Further more, such problems can be of different nature: either a failure of the system (endogenous) or a user that misused the IT system (exogenous). In the latter case, it is necessary to define whether the misuse is due to the user who should not have performed the failure action or if the IT infrastructure does not fit the business requirements.

1.1.1.2 IT Governance Objectives

Hence, IT governance (Definition 1), inspired from corporate governance, has been set up in companies and organizations to tackle the problems described in the previous section. Considering the importance of the sovereign and economic risks related to IT, many governments have defined informal and formal guidance in order to minimize these risks. Some governments have even defined legislative regulations: the US Sarbanes Oxley (SOx) mandates specific controls around IT to control the investments of organizations in USA and banks operating in Turkey must use the CobiT framework to manage their IT.

Definition 1: IT Governance (Guldentops et al., 2003)
IT governance, like other governance subjects, is the responsibility of the board of directors and executives. It is not an isolated discipline or activity, but rather is integral to enterprise governance. It consists of the leadership and organizational structures and processes that ensure that the enterprise's IT sustains and extends the enterprise's strategies and objectives.

The goal of IT governance is to define the organizational responsibilities in the IT's processes and how these processes support the business strategies and objectives. Namely IT Governance allows to ensure the alignment of the IT with the business activity, to exploit opportunities and maximize the benefits, to ensure that the use of the IT resources is effective and allows an appropriate management of IT-related risks. Thus, the CobiT framework identifies five focus areas: (i) *Strategic Alignment* to ensure the alignment of IT operations with the business ones by defining and validating the IT value proposition, (ii) *Value Delivery* that executes the proposition concentrating on the cost optimization, (iii) *Resource Management* to determine the investments and optimizes the IT resources, (iv) *Risk Management* that addresses the safeguarding, the disaster recovery and continuity operations, and (v) *Performance Measurement* to track and monitor the strategy implementation through all the steps of the governance process.

These five focus area guide the governance process, described in the next section, in order to insure its efficiency with respect to the business objectives fulfillment and the control of the IT infrastructure and resources.

1.1.1.3 IT Governance Workflow

To achieve these objectives, the CobiT framework provides good practices and a process framework to support the governance of IT systems. The good practices are consensus of experts based on experiences reported in workshops and/or widely spread. CobiT processes are defined from the business goals ensuring the *strategic alignment* objective. Business goals are turned into three dimensions of IT objectives following the three following steps: (i) *Business Goals* are used to define a set of *IT Goals* which (ii) set the *Process Goals* and (iii) themselves are used to define *Activity Goals*.

Figure 1.1a describes such top-down cascading goals definition illustrated by Figure 1.1b through the example of a security requirement. Along these three IT governance dimensions, metrics are set to evaluate the achievement of the governance objectives. A control cycle ensures the governance realization, represented in Figure 1.1a. This control cycle is divided into four steps, represented in the figure by the side arrows, starting from the top-left corner as follows :

1. **Goals definition** along the three governance dimensions as described previously (business, process and activity);
2. **Achievement measurement** along the three dimensions, using different levels of metrics;
3. **Performance indication** are established based on the previous measures. The lower is the dimension the easiest is the metric to measure. So outcome measures are used as performance indicators for the upper dimension –Figure 1.1b shows a typical example of such metrics: the frequency of review of security events to be monitored (*Activity Metric*) denotes a symptom of the number of incidents of unauthorized access (*Process Metric)* which helps in turn to estimate the upper metrics;
4. **Improvement and Refinement** corrects and optimizes the governance, according the performance evaluation, to improve the IT.

It appears that the evaluation of the governance performance is a an essential step in order to validate and improve the IT governance. The next section describes how the governance can be evaluated.

1.1.1.4 IT Governance Evaluation

In the CobiT framework, the IT governance is evaluated using benchmarks. In this way, the IT governance is rated following a scale of *maturity model.* Table 1.1 gives a representation of such a ranking scale.

Nevertheless, IT governance is still hardly evaluable formally. Indeed, while this maturity model is inspired by the *Capability Maturity Model (CMM)*[4], developed by the Carnegie Mellon Institute for *software development processes* evaluation, it has a softer definition. In fact, process are rated by a profile. Thus, its maturity can be considered to be mainly at the *Defined Process* level but still have some lower requirements compliance and yet being compliant with upper requirements.

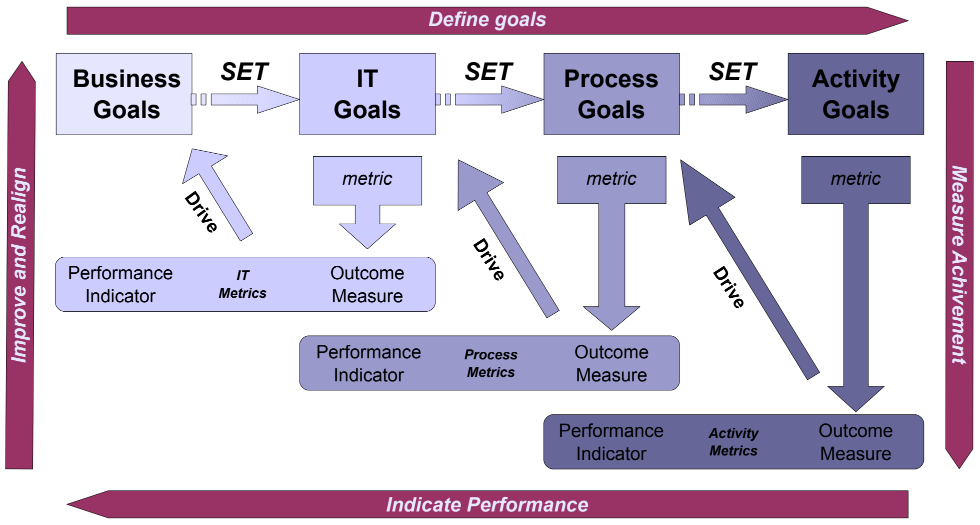
Such a maturity profile allows a finer evaluation than the classical CMM, but it also reveals the limits of such a governance model. In particular, it requires the analysis of an large number of experiences and case studies. This is hardly reproducible in the context of M2M governance, which is not yet developed enough to provide such an amount of experiences.

Furthermore, the lack of experience is not the sole obstacle for applying IT governance principles to M2M governance.

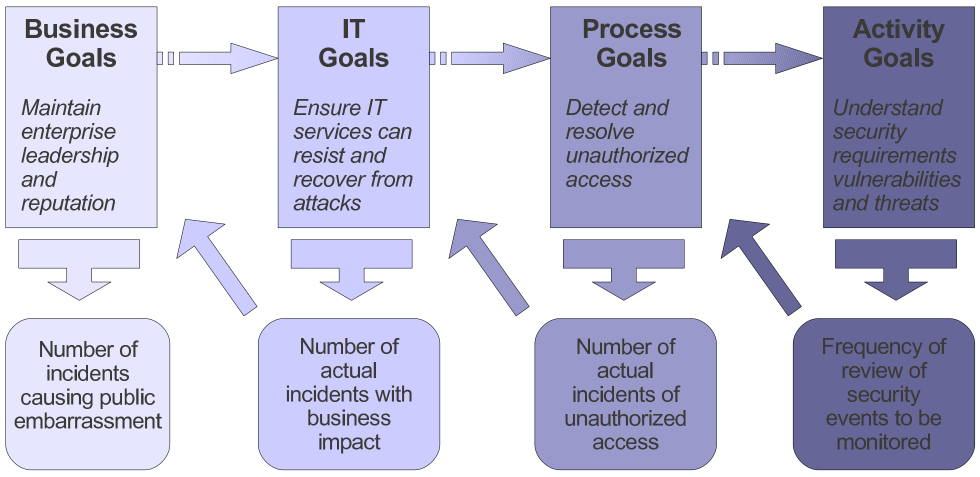
1.1.1.5 IT Governance limits

The governance is the matter of the board of directors. This ensures the alignment of the IT with the enterprise objectives. But, de Oliveira Luna et al. (2010) argue that such top-down approach also lacks of responsiveness. In fact, such a governance cycle and benchmarking the governance's processes require a lot

[4]The Capability Maturity Model Integration (CMM-I) Institute is a reference set of criteria to evaluate software providers. See `http://cmmiinstitute.com/`



(a) Goals, Processes and Metrics in CobiT


Business Goals
Maintain enterprise leadership and reputation
IT Goals
Ensure IT services can resist and recover from attacks
Process Goals
Detect and resolve unauthorized access
Activity Goals
Understand security requirements vulnerabilities and threats
Number of incidents causing public embarrassment
Number of actual incidents with business impact
Number of actual incidents of unauthorized access
Frequency of review of security events to be monitored

(b) Example of Goals and Metrics

Figure 1.1: Relationships among CobiT goals, processes and metrics (Figure 1.1a) and example of goals definition and outcomes metrics for a security requirement (Figure 1.1b).

of time for the evaluation. Hence, de Oliveira Luna et al. (2010) defend an agile governance model focusing on the customers' requirements in order to make the governance evolve quicker.

If the complexity of the governance's evaluation increases this lack of reactivity, this is even more complex in open organizations like network organizations. According to Kai et al. (2009), *"a* Network Organization *consists of several autonomous organizations that behave as a single larger entity in a long-term collaboration"*. Such a definition includes IT systems that involve several company divisions, outsourcing companies, or partnership with external companies. In fact, despite their autonomy, all of these entities are involved in the same processes which *"create inter-dependencies that rest on the entangling of obligations,*

Rank	Level	Description
0	Non-existent	No management process applied
1	Ad Hoc	Ad Hoc and disorganized processes
2	Repeatable but Intuitive	Regular patterns followed
3	Defined Process	Documented processes
4	Managed and Measurable	Monitored and measured processes
5	Optimized	Good practices automated

Table 1.1: Maturity Models ranking in CobiT (Hayes et al., 2007).

expectations, reputations and mutual interests".

In particular, M2M systems can be considered as an *open networked organization*. In fact, such systems involve many stakeholders collaborating together to share the IT infrastructure and resources (see section 1.2). In order to deal with such open systems, the SOA paradigm brings new solutions for the collaboration of autonomous open IT systems. As SOA has become a *de facto* standard, we provide, in the next section, an overview of such an approach and the related governance principles.

1.1.2 Service Oriented Architecture (SOA) Governance

According to Brown and Cantor (2006), the Service Oriented Architecture (SOA) was developed in order to achieve better alignment between the business and the IT worlds. The SOA aims at improving the responsiveness and flexibility of the IT the formation of the IT into a set of services and to achieve the goals by a flexible composition of these services.

1.1.2.1 Service Oriented Architecture

According to the OASIS consortium[5] SOA is based on two key concepts: *service* –or Web Service (WS)[6]– and *composition*.

- A **service** is *"a mechanism to enable access to one or more capabilities, [provided] using a prescribed interface. [...] A service is opaque in that its implementation is typically hidden from the service consumer except for (i) the information and behavior models exposed through the service interface, and (ii) the information required by service consumers to determine whether a given service is appropriate for their needs. "*
  Hence, different service providers can propose their services to the users who can choose them depending on their needs. This allows a limited investment as the service is reused by several clients,

[5]The Organization for the Advancement of Structured Information Standards (OASIS) Consortium drives the standardization and the convergence of Web Services and e-business initiatives.
See `https://www.oasis-open.org/`

[6]The concept of service is independent from its Web Service (WS) implementation but it is often used in the same sense in the literature. As this difference is not relevant for our purpose, we will use both the terms *Service* and *Web Service (WS)* in the same manner.

but must of all, it transforms the way investments are done in the IT: services can be billed depending on their use. And if a service fails, it can be replaced by another provider.

- **Composition** is the operation through which a service directly use one or more services to provide its own service. Thus, it follows the dependencies between the different entities of the IT. Identifying the different tasks and sub-tasks of a process helps to define simpler services. Then, a complex service can be defined from the composition of the other services. Further more, the simpler the services are, the easiest they are to be developed, maintained and duplicated.

Following these principles, two main actors are involved in a SOA: (i) the *service provider* and (ii) the *client* – sometimes (iii) a *broker* is involved to match the provider with the service requested by the client. There is a *contract* –implicitly or explicitly defined– between these actors when the client requests the service. Furthermore, a provider can delegate some processes to other providers, as in corporate outsourcing, by the mean of service composition, and so become the client of another provider.

In the remainder of this manuscript, we will refer to the different actors involved in such an open system as *stakeholders* to express the fact that they may belong to different organization, having their own interests and objectives, and so, be governed by different policies. In this context, the contract is expresses a common governance for all of the stakeholders involved.

Thus, SOA brings new issues regarding the governance concerns: (a) SOA are naturally open IT systems, which (b) involve several stakeholders participating in (c) an implicit or explicit contract that (d) expresses the common objectives and governance rules.

1.1.2.2 SOA Governance

As described in the previous section, SOA raises new governance issues and make governance more challenging (Brown and Cantor, 2006). In fact, service composition is distributed in nature and these different services can be maintained by different organizations. Therefore, SOA governance must bring new features to the classical IT governance. In particular, the definition of the concept *Service Level Agreement (SLA)* adds the notion of contract to the traditional functional and non-functional requirements definition.

SOA policy rules, decisions and enforcement should have a particular attention together with the lifecycle management of the service portfolio. An SOA lifecycle management is given by Brown and Cantor (2006) that define four phases: (i) the *model phase* consists of defining the business requirements and designing the services, (ii) the *assemble phase* integrates IT artifacts within the processes, (iii) the *deploy phase* resolves the application's resources dependencies and hosting environments and (iv) the *manage phase* focuses on the maintenance of the operational environment and policies based on the performance monitoring and eventually update the business model.

In Brown and Cantor (2006), the IBM Rational Center has defined a governance framework for SOAs. This governance is build following a four step lifecycle that allows a continuous governance process improvement. Such a lifecycle management follows concepts similar to the CobiT lifecycle management, described in Section 1.1.1.3, expressed in a different manner. The objectives and processes of the following four governance steps are detailed in Table 1.2:

1. Plan the governance requirements,

2. Define the governance approach,
3. Enable the governance model and
4. Measure and refine the governance process.

In the three following sections, we focus on different aspects of SOA governance. In Section 1.1.2.3, we compare the two main SOA approaches to reveal the importance of SOA governance. Then, we describe how SOA governance can be effectively defined through the concept of SLA, in Section 1.1.2.4, and, in Section 1.1.2.5, how the SOA can be monitored to check its compliance with such an SLA.

1.1.2.3 SOA Governance support

In order to make the SOA governance effective, several tools can be used. These tools are enablers that helps to enforce and ensure the consistency of the governance in its different aspects (Brown and Cantor, 2006): transport and protocols, security, service composition, monitoring and requirements management.

Two different approaches are used to implement Web Services: Simple Object Access Protocol (SOAP) and REpresentational State Transfer (REST). We give a brief description of these approaches with respect to the governance perspective.

- REST (Fielding, 2000) is not a proper protocol, it is an architectural model similar to the World Wide Web. It focuses on resources which are accessible through a Unique Resource Identifier (URI) and can be cached within proxy servers.

  The architectural design of REST have made it lightweight, simple dynamic and highly scalable. Therefore, REST is very popular: according to Vitvar et al. (2012), it represents 75% of the web applications APIs.

Step	Process	Objectives
1. Plan governance requirements	Understand the governance structures and environment Create an IT governance baseline Define the scope of the governance model Conduct change-readiness surveys	*Aligned IT wrt. business* *Governance strategy* *Governance measures*
2. Define the governance approach	Define and refine the governance processes Define organizational change Define IT changes in SOA development processes	*Detailed governance plan* *Process prioritized* *Defined rights, policies and measures*
3. Enable the governance model	Implement the transition plan Initiate SOA organizational change Implement the infrastructure for SOA	*Solution rolled out* *Automated workflows* *Collection and report put in place*
4. Measure and refine the governance	Measure the effectiveness of governance processes Measure the effectiveness of organizational change Refine the development and operational environments	*Governance executed and tuned* *Metrics gathered for later governance refinements*

Table 1.2: The SOA governance lifecycle (from Brown and Cantor (2006)
.

- SOAP (Mitra, 2003) is a protocol based on XML for transmitting messages (i.e. requests and responses). This verbose description of the services focuses on the application logic but can make it slower and longer to deploy.

  But SOAP-based architectures offer many tools to apply the governance objectives and strategy and to control the infrastructure's compliance. In particular, SOAP allows a formal description of the services using Web Services Description Language (WSDL). WSDL (Christensen et al., 2001) is an XML format to describe Web Services and relative information like data types. It is even possible to generate a WSDL document from a SOAP source code and vice versa.

  Further more, the service composition can also be formally describe using tools such as *orchestration*, supported by the OASIS with the BPEL language (OASIS, 2007), and *choreography* (Barreto et al., 2005). Thus, the SOA-based application can be described in flexible and modular manner.

  All of these tools are declarative languages which ease the specification and the understanding for human managers but also for software applications. In particular, it facilitate the definition and the control of Service Level Agreement contracts between the service providers and the clients.

  Therefore, as SOAP grants better guaranties to achieve a good governance compliance.

Even though some work investigate governance tools for REST architecture, Vitvar et al. (2012) note that REST still offers a poor support for SOA Governance. As a consequence, most enterprises still prefer to build SOAP-based IT (Brown and Cantor, 2006; Vitvar et al., 2012).

The two next sections focus on tools used to apply SOA governance.

1.1.2.4 Service Level Agreement (SLA) Management

Service Level Agreement (SLA) makes the governance requirements explicit for the system, allowing the requirement defined in *step 1* to be enabled in *steps 3 and 4* of the governance process (see Table 1.2).

The SLA represent an explicit contract between the service customer and the provider. An SLA is usually translated into Service Level Objectives (SLOs) which details the criteria to satisfy the SLA and how they are measured. Such measures are commonly known as Quality of Service (QoS), that is, a minimal/-maximal bound of a metric concerning the service.

IBM has developed the `WSLA` framework (Ludwig et al., 2003). It is a runtime architecture which contains SLA monitoring services. The `WSLA` language is an extension of the WSDL XML language. It allows to define the metric and the source of the measures as well as their frequency. Listing 1.1 gives an example of an *OverUtilization* SLA (Line 1) written with the `WSLA` language and the definition of the corresponding *PercentOverUtilized* metric (Line 9).

The main advantage of such a declarative approach are (i) a natural way for human to express the requirements, and (ii) computer readable tags. This allows for a simple, yet broad and powerful, interface between the IT staff and the SLA management software tools.

1.1.2.5 SOA Monitoring and Control

In this section, we review some of the tools that can be used to monitor the SOA and control it to fulfill the requirements.

```xml
<SLAParameter name="OverUtilization" type="float" unit="Percentage">
 <Metric>PercentOverUtilized</Metric>
 <Communication>
  <Source>YMeasurement</Source> <Pull>ZAuditing</Pull>
  <Push>ZAuditing</Push>
 </Communication>
</SLAParameter>

<Metric name="PercentOverUtilized" type="float" unit="Percentage">
 <Source>YMeasurement</Source>
 <Function xsi:type="PercentageGreaterThanThreshold" resultType="float">
  <Schedule>BusinessDay</Schedule><!-- defined separately -->
  <Metric>UtilizationTimeSeries</Metric><!-- defined separately -->
  <Value>
   <LongScalar>0.8</LongScalar> <!-- 80% -->
  </Value>
 </Function>
</Metric>
```

Listing 1.1: Example of an SLA defined with `WSLA`

Monitoring and control is a key part of the governance in order to evaluate its performances and effectiveness. Brown and Cantor (2006) highlight two aspects to focus: (i) the infrastructure itself and (ii) the services exposed.

- The main approach use distributed agents designed to manage and monitor each parts of the system (Brown and Cantor, 2006) based on the autonomic computing principles. These autonomic managers are build following the MAPE-K control-loop (IBM, 2006), see Section 2.2 for more details). Autonomic Computing ensure self-* properties to the infrastructure itself. Maurel (2010) shows that autonomic components are an effective technology for managing SOAs.

- Souza et al. (2011) propose an "Awareness Requirement" to monitor the infrastructure. This requirement refers to the others' success or failure to adapt the infrastructure. To achieve this it uses the Goal Oriented Requirement Engineering (GORE) approach: the functional requirements are represented by a goal decomposition connected to the tasks (Yu (1997), see Section 3.2.1).

- Herssens et al. (2010) use a Norm-Oriented Multi-Agent System to support the definition, management, and control of SLAs. Norms are used to express the SLOs. Normative agents represent the client and provider of the service. They fulfill roles corresponding to the contract agreement. Then an authority agent control their compliance with the normative obligations and apply sanctions in case of violation (See Section 3.2 for more details).

These tools enable to cycle the governance process by continuously adapting the SOA to meet the requirements. But, they also allow the IT staff to adapt the requirements, to meet new strategic objectives.

1.1.3 Synthesis on IT and SOA Governance

As shown previously, governance is a process to enable the IT system to meet the business objective. In both classical IT and SOA systems, such a process includes four steps:

1. the requirement definition according to the business needs,
2. the definition of complementary intermediate goals and processes to satisfy the previous ones,
3. the monitoring and control of the infrastructure to ensure the requirement validation,
4. the refinement of the governance objectives based on the performance measurement of the system.

Further more, open infrastructures, such as SOA, are more complex than traditional IT systems. As a consequence, the governance principles have evolved to face the new outcomes. In particular, new tools are used to enable a decentralized control and more adaptation. These tools enable an explicit definition of requirements using declarative languages, such as Web Services Level Agreement (WSLA), to facilitate the interaction between the human and the software controller, and so it reduces the delay for the whole governance process to cycle.

In the next section, we analyze the Machine-to-Machine (M2M) infrastructure in order to formulate the M2M governance specific needs.

1.2 Machine-to-Machine (M2M) Governance

As shown in the previous section, Governance is of great importance in the IT domain. It prevents from misuses of the IT resources and guarantees the IT alignment with the business objectives. Thanks to this approach, governance helps the organizations to reduce their infrastructure costs, define correctly the outsourcing policy and maximize the IT benefits.

Therefore, we now study the specific constraints of M2M systems and review the approaches contributing to the governance of M2M systems.

M2M is a key technology for achieving the Internet of Things (Estrin et al., 2000): connecting physical devices (sensors and actuators) to software services via the Internet in order to build ubiquitous applications. That is to say, the devices provide services to software applications which allow them to monitor and act on the physical environment. Recent improvements of Wireless technologies make it possible to deploy large networks of such devices at low cost (Atzori et al., 2010). As a consequence, it is now possible to think about large scale deployment of such technologies. A typical example of large scale application is the city: hence *Smart City* project are getting more and more common (Dirks and Keeling, 2009; Foschini et al., 2011). But, the deployment of such an infrastructure in the context of a Smart City raises several issues including (i) the heterogeneity and (ii) the number of the devices, (iii) the devices' energy constrains management, (iv) the number of stakeholders and (v) the heterogeneous requirements coming from (vi) a large number of M2M applications (Madhusudan and Bottaro, 2009).

As we saw in the previous section, Governance helps to manage efficiently the infrastructure and to maximize the profit from it. Therefore, in this section, we address the problem of the governance of the M2M. First, Section 1.2.1 describes the M2M architecture and draws issues arising from such an architecture. Then, Section 1.2.3 reviews the approaches which address these issues.

1.2.1 Key Technologies and Infrastructure Design of M2M systems

Machine-to-Machine (M2M) is an early technology which is just raising out of fully proprietary solutions with different standard proposals. M2M is principally based on two key technologies and two architectural designs:

1. *Low Power and Lossy Networks (LLNs)* technologies for communications between sensor and actuator devices, associated with
2. *cellular networks* providing a broadband connection to the Internet. On top of these technologies, two key architectural designs allows large scale M2M deployments:
3. *N-tier achitecture* to abstract the access to the devices as *Web Services* for the M2M applications –which deliver value added services to the end users–
4. a division between *horizontal* and *vertical* concerns.

In the following four sections, we describe and these four technologies and architectural designs, before drawing the Governance issues in Section 1.2.2.

1.2.1.1 Low Power and Lossy Networks (LLNs)

First, M2M is based on **wireless radio technologies**. The IPv6 Low power Wireless Personal Area Network (6LoWPAN) standard defined by the Internet Engineering Task Force (IETF) enables to use IP routing over the `802.15.4` communication protocol (defined by the IEEE Wireless Personal Area Network Task Group 4 (IEEE WPAN TG4)). These standards allows low power wireless communications on the 2.4GHz bandwidth to build so-called Low Power and Lossy Networks (LLNs) (Ma and Luo, 2008). Thus small messages can be send by/to devices such as sensors or actuators on broader distances than bluetooth and using less power than wifi (Akyildiz et al., 2002; Atzori et al., 2010). Such a network is called a Wireless Sensor and Actuator Network (WSAN).

Thanks to such LLNs technologies, very constraint devices can be large scale deployed (Madhusudan and Bottaro, 2009) at a very low cost and with low power supply. Thus, a devices can be supplied by a single battery during several years. Nevertheless, the main drawbacks of such technologies are (i) high loss rate, (ii) high latency and (iii) low transmission capacity (only few KBytes/sec).

1.2.1.2 Cellular communication

An other key enabler of M2M infrastructure is the use of **cellular networks** as a broadband connection (Martsola et al., 2005). In fact, given the constrains of LLNs, the devices cannot be directly connected to the Internet.

Then, these devices can the be connected to the Internet using a broadband Gateway. This broadband connection can be wired. For example, the French Electricity Provider (EDF) proposes to use a PLC and Optical connections to connect its `Linky` electricity meter[7].

[7]Linky website: `http://www.erdfdistribution.fr/Linky/`

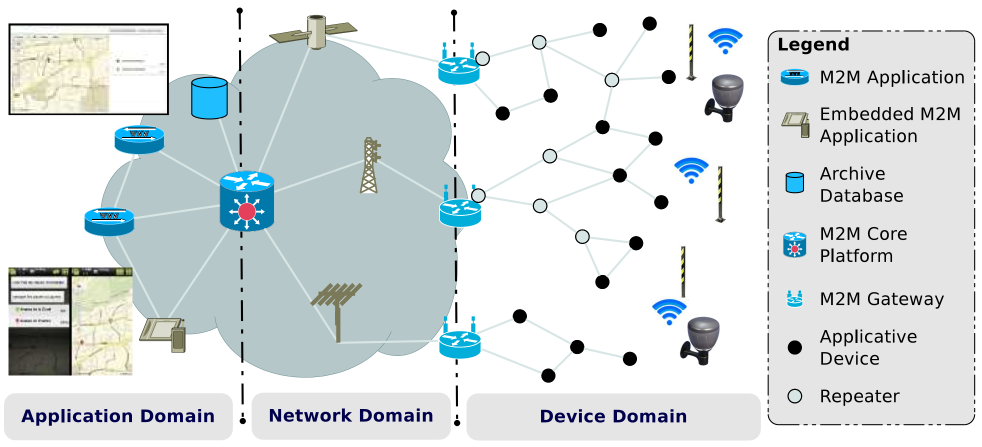


Figure 1.2: An end-to-end M2M architecture divided into three domains (from the right to the left): Device, Network and Application.

Even though, a cellular connection is more flexible. It allows to place the Gateway at the appropriate position regarding its WSAN coverage (Barthel et al., 2010). Nevertheless, cellular networks have a lower capacity than wired (e.g. xDSL, Optical) networks and suffer overload congestion due to the mobile traffic. Therefore, Foschini et al. (2011) state that the M2M gateway should be offline as most as possible which can be achieved using existing Telcos' *Infrastructure Management Service*.

This makes the Telecommunication Operators (Telcos) a strategic stakeholder for delivering the access to the sensor devices (Foschini et al., 2011; Morrish, 2012).

1.2.1.3 M2M N-tier architecture

Gold et al. (2008) show that most of the M2M application are designed using the *publish/subscribe* messaging paradigm: (i) a client application subscribes to the devices' services and (ii) the devices sends their data to the subscribers. Using an SOA approach, such mechanism can be divided into several parts (Spiess et al., 2009). Typically, the M2M infrastructures use a **N-tier architecture** to control the devices (Ückelmann et al., 2011): (i) the M2M application sends and receives messages to/from the devices via (ii) a web server which routes these messages to/from (iii) the Gateway which in turn (iv) redirects them to/from the devices.

The main benefit from the N-tier architecture is its flexibility. In fact, it allows to separate the applicative concerns from the device management concerns. Figure 1.2 describes an end-to-end M2M N-tier architecture. This architecture is divided into three domains corresponding to different concerns: (i) Device, (ii) Network and (iii) Application.

The Device domain is composed of applicative devices –i.e. sensors and actuators– but also repeaters used to broaden the WSAN linked to a gateway. The gateway manages one or several WSANs: security and device authentication, but also quick reaction to messages sent by the devices. Gateways also links the Device domain with the Network domain.

The Network domain is about routing messages between the Gateway and the M2M Core Platform, gen-

Stakeholders	Roles
Service Provider	Provides value-added services to the end users using the M2M devices
Device Supplier	Manufactures M2M devices, ensures their maintenance and provides the integration tools
LLNs Expert	Plans the deployment of the M2M devices
Telcos	Administrates and manages a Wide Area Network (WAN)

Table 1.3: Roles of the different M2M stakeholders.

erally using a broadband technology: cellular, satellite, xDSL or PLC networks. The Core Platform consists of one or several server in charge of brokering and managing requests from applications and the communication with the devices.

The Application domain is the part of the M2M infrastructure which runs the business logic. M2M applications send requests to the Core Platform to subscribe. These applications can provide ubiquitous services using the data collected from the sensors or using the users requests to act on the physical environment through actuators.

For example, a Smart City application, such as telemetering, would be deployed within hundreds of thousands houses. In order to transmit the sensor data to the gateway, repeaters must be deployed and efficiently situated. Then the data are collected to a server which stores and delivers it to the applicative servers.

1.2.1.4 Vertical versus Horizontal M2M deployment

In the industry, the term *vertical deployment* refers to an M2M infrastructure where all the parts are deployed for one application by a single stakeholder (Morrish, 2012). Nevertheless, building such *vertical* solutions is not flexible enough and is too expensive to be deployed large scale (Madhusudan and Bottaro, 2009).

Thanks to the N-tier architecture, the different concerns can be shared between different stakeholders (ETSI, 2011; Madhusudan and Bottaro, 2009; Santucci, 2008). Table 1.3 describes the roles corresponding to the different stakeholders: service providers, devices constructors, LLNs radio experts and Telecommunication Operators (Telcos). When such a deployment is fully shared between different stakeholders, we call it an *horizontal deployment*.

As a consequence, there is a growing need and interest for M2M infrastructures that provide an *horizontal* integration and sharing of devices between the stakeholders (Barkai, 2008). Considering the previous example, different stakeholders would be responsible for the different parts of the M2M system (i.e. electricity meters, collection network, applicative servers). Thus, if other Smart City applications are deployed they can benefit from the network infrastructures and WSAN devices already set up.

It is important to note the difference between *vertical* and *horizontal* deployment of M2M (see Figure 1.3 for a graphical representation of vertical and horizontal deployments). Both of them have benefits and drawbacks.

While the former is dedicated to a single application, the owner have a better control on the infrastructure and strategic choice. But, it also increases the deployment and management costs.

On the contrary, the horizontal design allows to share the costs and responsibilities among several stakeholders. A better reuse of the infrastructure and M2M devices and data can be achieved. Nevertheless, it increases the complexity of the infrastructure management. In fact, the stakeholders must find a common agreement about the use of the resources, the access to the devices and data.

1.2.2 M2M Governance Issues

As M2M infrastructure are based on SOA, typical SOA issues, such as service openness and composition, also concern M2M governance. In addition, the nature of the M2M devices introduces new specific issues. Finally, the design horizontal deployments increases the complexity of the system and makes governance more critical.

In the following sections, we focus on three issues that are specific to M2M and/or more critical than other SOA:

1. Device life-cycle management
2. Heterogeneity of application requirements in horizontal deployments
3. Scalability of the M2M infrastructure

1.2.2.1 M2M Device Management

First, M2M networks can use heterogeneous devices: applicative devices (i.e. sensors and actuators) but also repeaters used to broaden the area covered by the WSAN. Each device can embed several sensors and actuators, giving them several capabilities. Further more, applicative devices might be used as repeaters too. Thus, the devices might be involved to ensure several functions.

Secondly, energy management is of first importance with respect to these devices. In fact, if the cost of such devices is low (Akyildiz et al., 2002) their deployment and maintenance operations is expensive as they requires human interventions. Therefore, Madhusudan and Bottaro (2009) report that these operations should be limited as most as possible. The deployed devices' lifetime is expected to be in a range between 10 and 20 years.

Such an issue is even more emphasized in the context of horizontal deployments. In fact, as a device could serve several client applications, it might be solicited more. In addition, the repeater devices have to handle more traffic and more routes to the business devices. This aspect is also related to the scalability issue.

1.2.2.2 Heterogeneity of Application Requirements in Horizontal Deployments

In horizontal deployments, we have to take into account the heterogeneity of the requirements from the different applications.

Application	Sensing Rate	Trans. Rate	Message Size	Max Loss	Max Latency
Water Metering	4/H	1/D	100B	$O(10^{-2})$	300s
	4/H	4/H	10B	$O(10^{-2})$	300s
Electricity Metering	1/5s	1/D	2KB	$O(10^{-2})$	300s
	1/5s	1/H	100B	$O(10^{-2})$	300s
Gas Metering/Switch	4/H	1/D	100B	$O(10^{-2})$	300s
	4/H	4/H	10B	$O(10^{-2})$	300s
(switch control)	–	1/Week	10B	$O(10^{-3})$	30s
Garbage Monitoring	N/A.	1/Week	N/A.	$O(10^{-3})$	30s
Pollution Alert	1/min	4/H	1KB – 5KB	$O(10^{-2})$	30s
	1/min	–	1KB – 5KB	$O(10^{-3})$	30s

Table 1.4: Example of Non-Functional Requirements for M2M applications.
(From Barthel et al. (2010))

For example, Barthel et al. (2010) describe different non-functional requirements as an applicative framework for the ARESA2 project[8] summarized in Table 1.4. It reveals that requirements may be different from one application to an other, but they also could vary depending on the context of the application.

This can raise even more problems as different stakeholders may have contradictory requirements or interests for the same set of devices. For example, one application could request a sensor value every morning, while another one could request it every evening. Such a case would double the load on the device, and so, shorten its lifetime.

Considering the constraints of the devices and the heterogeneity of the requirements leads to a lack of confidence in horizontal deployments –i.e. stakeholders are reluctant to share their devices with others.

It is the case, for example, in the SensCity project[9], driven by Orange Labs in Grenoble, which goal is an "horizontal" deployment of an N-tier M2M infrastructure. In fact, all the partners agreed to share the network and server infrastructures to deploy their own sensors and devices. But a mandatory requirement was to ensure them a monopolistic access to the devices and data for their own M2M services. Despite the economic aspect of this experiment –i.e. keep the data and sell the value-added service–, the cause of such a requirement is the fear the devices would collapse too quickly. This "multi-vertical" view considers the shared infrastructure as a third-party platform for the integration of multiple vertical deployments (Morrish, 2012).

Figure 1.3 summarizes the three types of M2M deployments: (a) each application deploys its own M2M infrastructure, including the M2M devices and network connections, (b) the network infrastructure is shared by several applications to access to their own devices, and (c) the network infrastructure is shared beyond all the stakeholders, such as the devices, so the applications can dynamically access to all the resources. To do so, the governance must guarantee the M2M devices providers that their devices' constraints will be respected by the other applications.

[8]http://aresa-project.insa-lyon.fr/

[9]SensCity official website available at: http://www.senscity-grenoble.com/

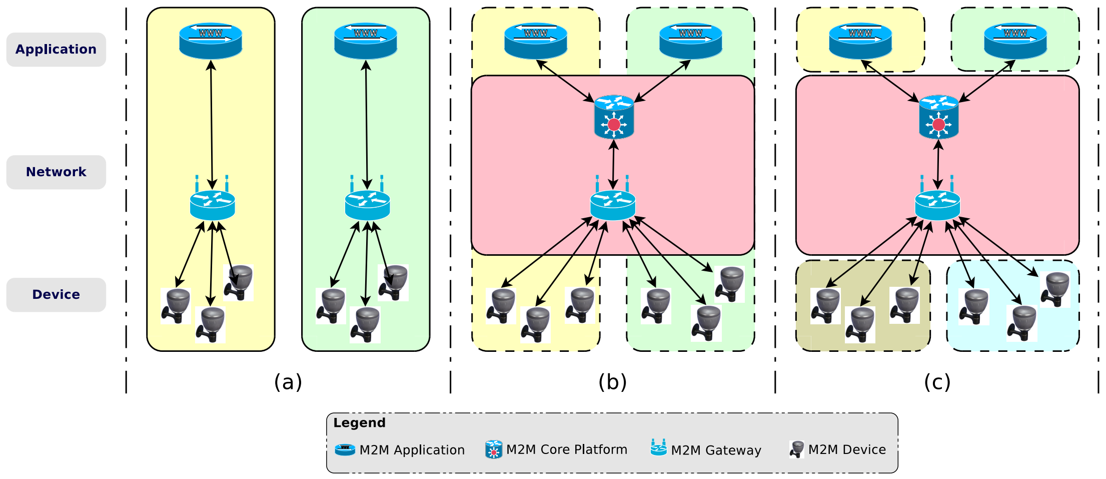


Figure 1.3: Representation of three types of M2M deployments: (a) two individual *vertical* deployments, (b) a *multi-vertical* deployment and (c) an *horizontal* deployment.

1.2.2.3 Scalability Management

Finally, all these issues are connected to the scalability issue. Economic trends predict 30 billions devices to be connected the Internet of Things within 20 years (OECD, 2012). In fact, the scale of *Smart Environments* applications is increasing: M2M projects include *Smart Home*, *Smart Office*, *Smart Building*, *Smart Hospital/Home Health Care*, *Smart Logistic*, and *Smart City*. Of course, this number includes passive RFID devices, which will not necessarily generate much traffic. Still, it means a large number of entities to take into account in database and process models.

But in the case of *Smart City* applications, infrastructures are expected to handle a large number of devices, used for many urban services and generating a high communication traffic. Table 1.4 describes only a few applications but can give us an idea of the traffic generated by such requirements if hundreds of thousands devices are deployed in a city. The requirements described in Table 1.4 shows that each sensor is expected to send a message from once a day to once every 4 hours. That is the repeater nodes, the gateways and the collect servers should handle a number of message proportional to the number of sensors. Hence, while scaling the number of devices, the load –and risks for overloads– increase on the network infrastructure.

The scalability issue is enforced when considering *horizontal* deployments. In fact, because of different applications' requirements, not only the infrastructure load would be impacted, but the sensors' load itself will be increased. The same sensors could be requested to send sensed data at the beginning of the day by an application and at the end of day by another one. Hence, the generated traffic is twice bigger and so is the computation load and the battery consumption of the devices. For example, in the ARESA use cases, electricity meters are supposed to sense the electricity consumption and to be used as repeaters for gas and water meters.

Therefore, it is of great importance to define the governance for such M2M systems. These governance should consider the following aspects of the system: (i) very constrained devices with respect to both

a) computation and b) energy capacities, (ii) limited access to the devices through LLNs, (iii) heterogeneous applicative requirements and (iv) the scalability of M2M deployments.

1.2.3 Approaches for governing M2M infrastructures

Different approaches in the litterature tackle the governance problem for M2M infrastructures, addressing one or several issues from a different point of view: principally (i) the management of the WSAN's devices, (ii) the applications' access to the devices and (iii) the M2M infrastructure's Functional and Non-Functional Requirements. In this section, we draw a synthesis of these approaches from a governance perspective. Based on this analysis, we try to highlight the missing features for an effective governance M2M infrastructures.

1.2.3.1 The M2M Platform: a key component of the M2M infrastructure

A survey of Wireless Sensor and Actuator Network (WSAN) management (Gold et al., 2008) reveals that in most of the WSAN frameworks the management of the M2M devices is delegated to an external resource/server. In fact, while several protocols and routing algorithms can be used at the Device Domain level to optimize the energy consumption or the scalability of the WSAN (Akyildiz et al., 2002; Anthony and Mccann, 2007; Iwanicki and van Steen, 2010; Erdene-Ochir et al., 2011), the devices' computation capacity limits their capacity to take into account the overall governance objectives. For example, Lampin et al. (2012) propose the *QoS oriented Opportunistic Routing (QOR) protocol* to deal with the trade-offs between possibly antagonistic QoS objectives: *delivery ratio*, *delivery delay* and *energy consumption*. Nevertheless, the metric used for these algorithms is static and specifically defined for this algorithm. Therefore, a change in the objectives would require a software/firmware update to be done externally.

Given the constrained nature of the devices, the key part of the M2M N-tier architecture is the *Collect Server(s)*. In fact, its central (but not necessary centralized) position in this type of architecture makes it strategically efficient to build a global overview of the system.

In fact, Herstad et al. (2009) claims that such a platform brings more flexibility and adaptability by facilitating the integration of independent devices and applications. It uses a modular platform so that basic modules mainly handles the communication between the other modules. Additional one can be used to support advanced operations such as End-User Management, SLAs, Life-Cycle Management...

1.2.3.2 Integration of the M2M within the Internet

Shelby (2010) proposes to extend the existing Internet architecture for WSAN. This work is driven by the IETF Constrained RESTful Environments Working Group (CoRE) standardization group which is working on the Constrained Application Protocol (CoAP)[10]. This protocol extends the REST protocol in which the messages are compressed to reduce their size. Hence, the M2M device are considered as WS providers.

The CoRE architecture is based on the assumption that the Internet has proved its ability to scale due to its decentralized architecture. In particular, the use of *proxy servers* makes possible to cache the sensors' data. Hence, responses to the requests to the sensors can be sent either by the sensors of by the proxy servers, based on their cached values.

[10]CoAP Draft Available on the IETF website: `https://datatracker.ietf.org/doc/draft-ietf-core-coap/`

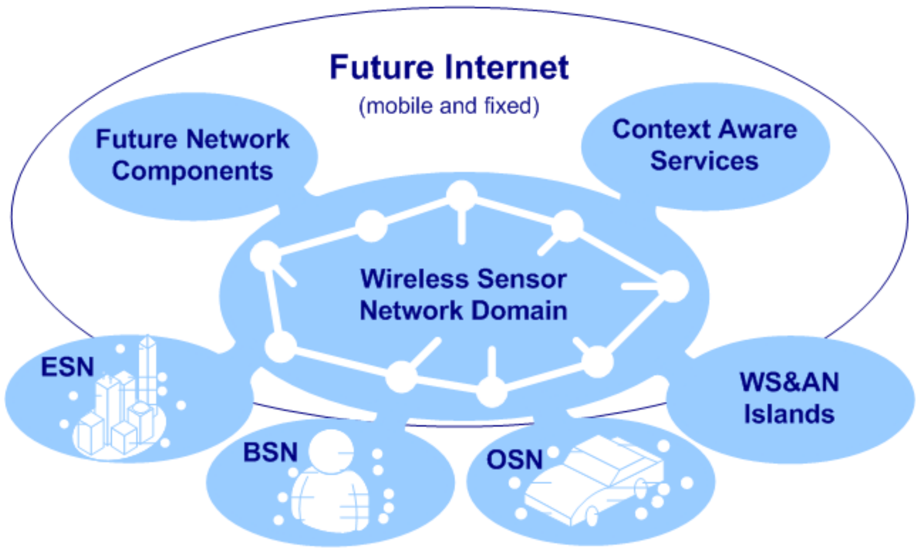


Figure 1.4: The SENSEI approach based on WSAN Islands.

Using proxy servers allows to virtualize the M2M devices reducing their load and energy consumption. In addition, simple management rules can be set up such as authentication and access rights, for example.

The SENSEI[11] project also virtualizes the representation of the devices in so-called *WSAN Islands* in the context of Smart City M2M infrastructure (see Figure 1.4). This WSAN Island enables to achieve the integration of heterogeneous WSAN into the same framework –i.e. different type of M2M devices, using different radio technologies and protocols. Each WSAN Island represents a set of physical M2M devices as a *Resource Endpoint*. It is identified by a normalized URI which integrates the M2M resource in the Internet. It is, then, accessible by any application which have the corresponding access rights.

Baugé and Bernat (2008) propose the Representation Endpoint to be divided in two parts: (i) a *basic resource description* which contains information such as the resource ID or the service description (e.g. `WSDL`), and (ii) an *advance resource description* which contains information such as a semantic description of its capabilities or its dynamics. The Resource Endpoint can be decoupled from the physical host of the resource, so that highly constrained devices with low computation capacity are represented on a higher performance server.

In order to manage the resources of a WSAN Island, special resources are used (*Management Resources*). The corresponding Resource Endpoint allows for classical security and access policy management for example. But a software agent can also be used for advance management functions: e.g. managing software/firmware updates, supporting and validating traffic pattern, supporting fault management, supporting context awareness management, . . . Thus, the Management Resource Endpoint can be used by the M2M devices' operators to adapt the behavior of their devices –i.e. manual reconfiguration.

The SENSEI project provides a framework for an end-to-end management of heterogeneous M2M resources. This framework allows to apply the Governance Decision thanks to the Management Resource Endpoint. Nevertheless, this framework does not explicitly define the Functional and Non-Functional Re-

[11]The SENSEI website: `http://www.ict-sensei.org/`

quirements of the overall infrastructure. While the framework's architecture provides mediation support between the WSAN Island, it does not support the mediation process definition. Though it is possible, for the operators, to automate the management of their own devices, the definition of the M2M Governance is out of the scope of this framework and is let to each M2M operator.

1.2.3.3 The M2M Infrastructure Requirements

The SOCRADES Integration Architecture (Spiess et al., 2009) provides an abstraction of the M2M Collect Platform using an SOA. Such an architecture allows to integrate the M2M platform within the enterprise's IT processes –e.g. classical ERP. The M2M Devices are accessible through a WS interface. A Service Catalog provides brokering to the devices and select the "best" suitable device in terms of QoS corresponding to the application's needs. Further more, it allows to use the SOA Governance principles and tools. But, as stated in Section 1.2.2, M2M architectures have more specific constrained to take into account in the governance.

The US Ocean Observatories Initiatives use a WS-based architecture to provide access to Ocean Sensor Platforms to Scientific Teams. Wefer, Gerold (2009) use a architecture for contract agreement between applications and sensor resources. The applications send the requests to a *Resource Planner* corresponding to the sensors. This Resource Planner is a solver based on a resource constraint model which express the Governance Policy for a sensor or a set of sensors. The result is a *Resource Use Plan* which can be considered as an SLA.

This work is of great interest from the *Vertical Governance* point of view. However, two main drawbacks come to us: (i) the platform's architecture is very specific to the scientific observation of the oceans, and more important, (ii) the *Resource Use Plan* is based on a model specific to the sensor requested. Therefore, the governance policy is not decoupled from the model of the sensor capabilities and energy consumption.

The European Telecommunications Standards Institute Technical Committee on M2M (ETSI TC M2M) aims at providing a standard architecture for M2M infrastructures. ETSI (2011) defines the Functional Architecture for three reference points: (i) the Core Network Platform (i.e. the collect server), (ii) the M2M Gateway and (iii) the M2M Devices (in the case the device is directly connected to the Core Platform).

Each of these reference points provides a list of *Service Capabilities* that are to be exposed. Each capability corresponds to one or several functionalities the M2M components should ensures independently from the applicative purposes. A complete description of these capabilities is given in Appendix C. A synthesis of the capabilities is given in Table 1.5. Considering the M2M Core Platform, which is at the frontier between the Network and Application domain, we can classify these capabilities depending on the M2M domain they address.

Although they could be considered as Non-Functional Requirements from the M2M Service Provider point of view –i.e. "verticals"–, these capabilities can be considered as the **Functional Requirements of the M2M "horizontal" infrastructure**: from *Communication Management* to the *Application Enablement*.

Hence, the ETSI TC M2M's recommendations appears to be a starting point towards a standard Governance of the M2M infrastructures. It should be combined with the management of vertical M2M requirements.

REM	Remote Entity Management	Manage performance information and updates of the Devices
GC	Generic Communication	Single contact point with Gateways/Devices
RAR	Reachability Addressing and Repository	Manage devices sets information and subscription
CS	Communication Selection	Network selection policies to reach Devices
IP	Interworking Proxy	Compliance with non ETSI Devices, Gateways and Platforms
SEC	Security	Manage authentication and data integrity
AE	Application Enablement	Single contact point with Applications
CB	Compensation Brokerage	Allow brokerage between Devices's services and Applications
TM	Transaction Management	Interworking of capabilities exposed in NSCL
HDR	History Data Retention	Persistence of data and messages history

Table 1.5: The M2M Core Platform's capabilities. (Defined in ETSI (2011))

1.3 M2M Scalability Governance

In Section 1.2.2, we have study the specific issues raised by the governance of M2M systems: (i) the Constrained Device Management, (ii) the Horizontal Management of Vertical Requirements and (iii) the Scalability Management. Scalability is one of the most mentioned issue in the M2M literature (OECD, 2012; Herstad et al., 2009; Ückelmann et al., 2011; Spiess et al., 2009).

Further more, the scalability issue is orthogonal to the two others. In fact, the M2M devices' constraints make the scale of the system even more critical, so is the horizontal deployment along many verticals and stakeholders. Therefore, in this section we focus on the *Scalability issue in the M2M from a Governance perspective*.

First, we seek for a comprehensive definition of "scalability". As most of the literature reviewed here comes from the complexity community, we adopt the formalism used by the community for our definition. Then, we review how it is possible to measure the scalability of a system. And finally, we consider the different governance approaches for scaling a system.

1.3.1 What is scalability ?

The meaning of *scalability* is difficult to define. It differs from a system to an other one. Hill (1990) raises the problem of an abusive usage of this term in computer science. Even though the meaning can be intuitive, Hill shows there is no strict definition suitable in the general context (Hill, 1990).An official definition of the term *Scalability*, given by JO (2003) and CTB (2006), considers it as *"the degree to which a computer application or component can be expanded in size, volume, or number of users, and continue to function properly"*. But such a definition seems to be poorly useful to design and govern scalable systems.

1.3.1.1 Scaling along Dimensions considering Properties

Van Steen et al. (1998) proposed a definition of *scalability* based on three parameters: the performance measures p, the workload parameter l and the resources r: a system is scalable iff p is constant when l and r grows proportionally. This definition considers the performance degradation tolerance to be explicitly bounded by the δ function (**P2**) and infrastructure costs by the γ function (**P3**). A system is said scalable to a finite bound a_{max} (**P1**) iff the three properties **P1**, **P2** and **P3** are hold, following Definition 2.

Definition 2: Scalability Evaluation (Van Steen et al., 1998)
Let $\gamma(\mathbf{a})$ be a function in attribute values **a** *returning values in the same cost unit as $Cost_A$. Further more, let $\delta(\mathbf{a})$ be a function in a returning values in the same performance unit as $Perf_A$ with $\forall \mathbf{a} :: \delta(\mathbf{a}) \geq 0$ and $\delta(\mathbf{a}_{ref}) = 0$. An application A is scalable in $Attr_i$ for values up to a maximum a_{max} with respect to γ and δ if the following properties hold:*
P1: *A can accommodate values $\mathbf{a}_{ref}[i] < a < a_{max}$*
P2: $\mathbf{a}_{ref}[i] < a < a_{max} \exists \mathbf{r} ::$
$Perf_A(\mathbf{a}_{ref}, \mathbf{r}_{ref}) - Perf_A(\mathbf{a}_{ref}\langle i : a\rangle, \mathbf{r}) \leq \delta(\mathbf{a}_{ref}\langle i : a\rangle, \mathbf{r})$
P3: $\mathbf{a}_{ref}[i] < a < a_{max} ::$
$Cost_A(\mathbf{a}_{ref}\langle i : a\rangle, \mathbf{r}_{min}(\mathbf{a}_{ref}\langle i : a\rangle)) \leq \gamma(\mathbf{a}_{ref}\langle i : a\rangle, \mathbf{r})$

This definition is used by the TripCom testlab (Moritsch, 2008) which is a framework methodology for evaluating the scalability of a system. This methodology consists of (i) defining the components performance models and (ii) the whole system performance model, then (iii) analyzing the interdependency between components and system performance to (iv) build performance functions as a formula or benchmarks. Finally, (v) the scalability of the system is determined following Definition 2.

One can observe two problems with this definition and methodology. First, in Moritsch (2008) it may be very difficult to establish the interdependence between each individual component's performance and the global performance for very large scale distributed system.

1.3.1.2 Scalability as a Process

Weinstock and Goodenough (2006) base their definition on the process to achieve scalability: *"Scalability is the ability to handle increased workloads by repeatedly applying a cost-effective strategy for extending a system's capacity"*. This definition presents scalability more like a dynamic process rather than a simple property/capacity of the system. As a consequence, the authors suggest to submit the system to a Scalability Audit. This audit tackles the four following questions: (i) how to identify the potential resource bottleneck, (ii) how to reveal incorrect assumptions about scaling strategies, (iii) what are the strengths/weakness of the common scaling strategies and (iv) how to Scalability Assurance.

Further more, the authors suggest that scalability is mainly a matter of trade-offs between the properties of the system to handle the increased workload. Contrary to Van Steen et al. (1998) which considers that the two properties (performance and cost) should evolve proportionally, Weinstock and Goodenough (2006) give a more flexible trade-offs definition which can vary depending on the properties taken into account.

1.3.1.3 Scalability as a Goal

A more general definition is given by Duboc et al. (2007): *"Scalability is the ability of a system to maintain the satisfaction of its quality goals to levels that are acceptable to its stakeholders when characteristics of the execution environment ("the world") and design ("the machine") vary over expected ranges."*

Variables are used to represent the application domain, the system design and its properties. The variables are defined using the GORE methodology. Functional and non-functional requirements are formulated by a tree of goals for the system. Some of these goals are annotated with *Quality Variables* and *Objective Functions*. Goals are a description of stakeholder needs, specified as properties the system must satisfy. GORE also allows to define assumption of the system. These assumptions are also annotated with *Quality Variables* to describe scaling dimensions.

Using such a goal definition of requirements enables to manage them through the cooperation of software agents. *Quality Variables* define measurable properties of the system and the *Objective Functions* the scalability acceptable bounds. Then agents can be used to *detect partial satisfaction of the requirements* and to explore alternative system proposals. Thus, the decentralization of such an approach eases the definition of the scalability of the system.

However this methodology suffers some limits among which: (i) there is no explicit way to define thresholds for *Objective Functions* and (ii) to handle the conflicts resulting from partial satisfaction of the objectives, then (iii) using assumptions for modeling scaling dimensions makes unclear how to deal with their variations over the time.

1.3.1.4 Proposed Definition: Scalability Governance

Considering the different definitions given previously, we can synthesize the previous definition with respect to the M2M concerns as in Table 1.6: (i) awareness of the domain –i.e. M2M– specificities, (ii) a definition of the scaling dimensions and (iii) the systems' properties, which can be (iv) adapted considering bearable thresholds, through (v) a continuous scalability governance process.

Based on this synthesis, we come with a definition taking into account theses principles in Definition 3. We consider a system as scalable if, when scaling up one several dimensions, the requirements on its properties are satisfied, or if a bearable adaptation of the requirements is found. This adaptation might be the result of a balanced trade-off between several properties.

For example, let us consider an M2M system which required properties are P_1 : "`End-to-end latency < 1 minute`" and P_2 : "`Sensing Frequency = 1 / hour`". Then, let us consider a deployment of new M2M devices, resulting in a scale up along the dimensions D_1 : "`Number of devices`", D_2 : "`Load on the M2M platform`". The consequences of such a scale up might provoke an overload on the M2M platform and a failure of the maintenance P_1. A possible trade-off consists of relaxing the P_2 constraint to handle the scaling and maintain P_1 (or vice versa).

Definitions	Domain Specific	Dimensions	Properties	Threshold	Continuous Process
Official	–	–	×	–	–
Hill (1990)	×	–	–	–	–
Van Steen et al. (1998)	×	resources	performance cost	proportional	–
Weinstock and Goodenough (2006)	×	×	×	×	audit
Duboc et al. (2007)	×	×	×	–	GORE

Table 1.6: Scalability Definition Synthesis

Definition 3: Scalability Governance

Considering a system S, let be:

- $\mathcal{D}_S : \langle D_1, ..., D_N \rangle \subseteq \mathcal{D}$ *a subset of scaling dimensions affecting* S
  each scaling dimension $(D_i, \leq_{D_i}) \in \mathcal{D}_S$ *is a well-ordered set of scales* $s_{(D_i,1)}, ..., s_{(D_i,N_{D_i})}$ *by the ordering function* $<_{D_i}$
- $\mathcal{S}_\mathbf{X} = \langle s_{(D_i,\mathbf{X})}, ..., s_{(D_N,\mathbf{X})} \rangle \in \langle D_1..D_N \rangle$ *a possible state of the system* S
  $\mathcal{S}_\mathbf{ref}$ *the reference state.*
- $\mathcal{P}_S : \langle P_1, ..., P_N \rangle \subseteq \mathcal{P}$ *a subset of properties to be hold by* S
- $r_{P_i} : \mathcal{S} \longrightarrow \{True|False\}$ *a requirement evaluation function for the property* $P_i \in \mathcal{P}_S$

The system S is scalable for the dimensions $\mathcal{D}_S$ to a finite bound $\mathcal{S}_\mathbf{bound}$ the following property holds:

PSCALABLE $\forall \mathcal{S}_\mathbf{X} \leq \mathcal{S}_\mathbf{bound}, \forall P_i \in \mathcal{P}_S, \exists P'_i \in \mathcal{P}_S, r_{P_i} \oplus r_{P'_i} = True$

where P'_i is a bearable adaptation of P_i

Such a scalability management should be manage during the governance cycle as expressed in Section 1.1.1.3 and Section 1.1.2.2. First, the requirement definition (step 1–2) should specify the objectives with respect to the system properties. Then, both the property and dimension metrics should be used for monitoring and controlling the system's infrastructure (step 3). Finally, requirements should be refined (step 4) regarding bearable property objectives.

In the two following sections, we focus on the metrics –i.e. dimensions and properties– for characterizing the scalability of M2M systems (Section 1.3.2) and the possible policies to handle it (1.3.3).

1.3.2 Scalability Metrics for M2M Governance

In this section, we focus on metrics used to characterize the system's scalability, with respect to Definition 3. Firesmith (2010) defines a metric as a measure of one of the system's characteristics. It is used to assign a value to a qualitative characteristic of the system. For example, Figure 1.5 These metrics provide a

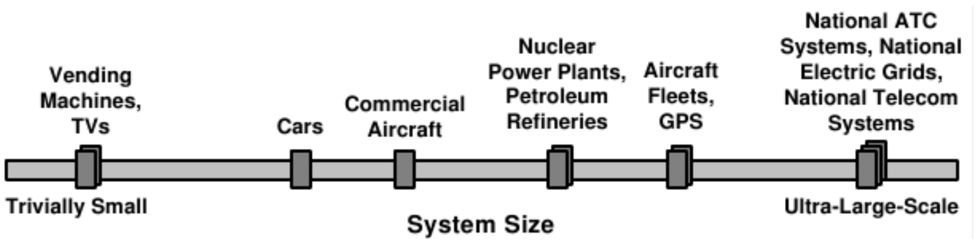

Vending Machines, TVs
Cars
Commercial Aircraft
Nuclear Power Plants, Petroleum Refineries
Aircraft Fleets, GPS
National ATC Systems, National Electric Grids, National Telecom Systems
Trivially Small
System Size
Ultra-Large-Scale

Figure 1.5: A metric used to describe the scale of several systems.

framework to describe a scale profile of Ultra Large Scale s (ULSs) and System of Systemss (SoSs).

In this section, we focus on the relevant metrics for profiling the scale of M2M systems. As stated in the previous section, we can consider two types of metrics: (i) dimensions and (ii) properties. *Dimensions* are metrics characterizing the state of the system while *properties* are metrics referring to the requirements of the system.

1.3.2.1 Scaling Dimensions

Size of the system

The first scale one might think of when talking about scalability is the *size of the system*. Nevertheless, this *metric* may vary from a system to another. In fact, we can consider several parameters related to the size of the system.

Estrin et al. (2000), Firesmith (2010) and Wilson et al. (2009) define the **number of devices** as one of the most important parameter as we consider M2M systems.

The **amount of data** is classical –and probably the most important– dimension considered the literature as reported by Bondi (2000), Michael et al. (2007), Neuman (1994) and Wilson et al. (2009). It is directly related to the previous parameter as the devices are an important source of data in M2M systems. Further more, the more the devices send data the more their lifetime will be reduced, especially for the devices supplied by a battery.

Another parameter, highlighted by Neuman (1994) and Weyuker and Avritzer (2002), is the **number of end-users**.

Finally, Firesmith (2010) state that it is also necessary to take into account the **number of sub-systems** and **stakeholders** as it will increase the management complexity.

Heterogeneity

The *heterogeneity* of the system and its sub-systems increases the difficulty for managing the system. It increases its complexity and can considerably reduce its performance. In fact, some intermediate systems might be required in order to ensure the interoperability between different systems.

Considering M2M systems, Akyildiz et al. (2002), Gold et al. (2008), and Madhusudan and Bottaro (2009) consider the **heterogeneity of the devices** as an important parameter to take into account.

Further more, for Neuman (1994) and Gold et al. (2008), this problem can be amplified by the **heterogeneity of the communication technologies** used by the devices. In fact, it requires intermediate devices to enable the communication between the different WSAN.

Another parameter is the **heterogeneity of the data** as it increases the complexity for exploiting the data (Neuman, 1994; Barkai, 2008). Related to this parameter, Firesmith (2010) the **heterogeneity of the software development** also complexes the exploitation of the M2M infrastructure. Gold et al. (2008) identify these two parameters as a key feature to be handled by the M2M platform.

Deployment Topology

M2M networks can be deployed in many different manners depending on the purpose of the system. As in classical network, this deployment impacts on the performance and the management of the system.

First, the importance of **geographical area** the system is spread in is pointed out by Bondi (2000), Firesmith (2010), Neuman (1994), Römer et al. (2002) and Gold et al. (2008). In fact, it is possible to give a hierarchy of network based on the area they cover: (i) Body Area Network, (ii) Local Area Network,(iii) Metropolitan Area Network,(iv) Wide Area Network,(v) National Area Network and(vi) Global Area Network.

Further more, the management such networks will vary depending on the **network topology** (Bondi, 2000; Römer et al., 2002): from centralized network, to structured networks or even ad-hoc networks.

An other aspect of network deployments is often highlighted in the literature (Firesmith, 2010; Neuman, 1994; Römer et al., 2002; Gold et al., 2008): the **distribution** of the data storage and applicative processes.

Finally, Akyildiz et al. (2002), Firesmith (2010), Römer et al. (2002) and Gold et al. (2008) consider the **mobility** of the devices as an important criteria of M2M systems. In fact, it impacts the topology management and the continuity of services provided.

Environmental Dynamics

As M2M devices are deployed in the physical world, the system is more sensible to the environmental dynamics than a classical IT system. But this dimension also includes the dynamics of the virtual environment.

As stated in Wilson et al. (2009), the **dynamics of the physical environment** itself have a big impact on the system. This parameters corresponds the events occurring on the system without any control on them. For example, more messages can be generated by the devices monitoring a specific event, affecting the overall architecture's performance. In Bondi (2009); Firesmith (2010); Neuman (1994); Weyuker and Avritzer (2002), this parameter is often taken into account in classical IT systems while considering external requests.

Römer et al. (2002) also note that the M2M devices, deployed in the physical environment, are faced to the problem of **noisy communications**. In fact, LLNs have an important rate of data loss. This problem impacts the quality of the data collected but also the lifetime of the devices.

Further more, Akyildiz et al. (2002), Estrin et al. (2000) and Madhusudan and Bottaro (2009) show that the lifetime of the devices is also impacted by the **hostility** of the environment. This hostility can result from the climatic conditions (implicit hostility) but also from degradations caused by people (explicit hostility).

Firesmith (2010) show that the dynamics of the system is correlated to its size and heterogeneity. This complexity has a consequence on the **emergence** of unexpected events, which can be positive but also negative. Among these events, Weinstock and Goodenough (2006) consider the **number of errors** as an important criteria to take into account.

1.3.2.2 Scalability Properties

Performance

The performance of the system is the main property the governance tries to improve. In fact, Moss (2011) reveals it is also the one customers and partners will focus on and pay for. It is often used to define the QoS required by the SLA agreements.

The main metric used for the performance of M2M system is the **latency** (Neuman, 1994; Weyuker and Avritzer, 2002). It corresponds to the time for an action to be achieved and is often measured in seconds. In M2M system, this metric is used at the device level (Lampin et al., 2012) and for the end-to-end transmissions (Barthel et al., 2010).

But Weinstock and Goodenough (2006) also identify the **efficacy** as an important parameter. It corresponds to the threshold of errors tolerated. For example, it can be the data loss or the measurements' margin of error.

Bondi (2000) and Firesmith (2010) show that the *efficiency* of the is of system is of great importance too. The system is expected to make an efficient usage of the resources. It is especially true in the context of M2M systems where the devices are very constrained (Akyildiz et al., 2002; Barthel et al., 2010).

Usability

Another property a scalable system should exhibit is the *usability*. In fact, the system is expected to keep being as usable as it was before scaling.

To this end, Estrin et al. (2000), Weinstock and Goodenough (2006) and Wilson et al. (2009) advise to keep a particular attention on the system's **availability**. Thus, it is necessary to state the system's accessibility in terms of time (e.g. asynchronous, any-time, real-time. . .) and space (e.g. access-points, mobility. . .).

Complimentary to the availability, the **security** is a prior focus, according to Neuman (1994) and Firesmith (2010). In the M2M context, it is even more difficult to guarantee the integrity of the WSAN due to the limited capacity of the M2M devices which can hardly perform classical security mechanism, such as encryption. Further more, Wilson et al. (2009) and Madhusudan and Bottaro (2009) extend the security parameter to the **safety** of the M2M devices which can be damaged.

The M2M and the Internet of Things (IoT) also raise the problem of the **privacy** (Madhusudan and Bottaro, 2009; Wilson et al., 2009; Ückelmann et al., 2011). In fact, the sensors and actuators are used by ubiquitous applications which generate and exploit information about individuals. For example, a parking assistant application tracks the users' moves and habits. This is emphasized in *horizontal* infrastructures which facilitate the overlapping of information.

Additionally, the system is expected to be **consistent** (Neuman, 1994; Weinstock and Goodenough, 2006; Weyuker and Avritzer, 2002). While the quantity of information might be too important to be centralized, leading to overloads, it is necessary to prevent contradictory distributed information and processes.

Firesmith (2010), Neuman (1994) and Wilson et al. (2009) identify another parameter of the usability: the **simplicity**. In the context of the M2M, the infrastructure should be easy to use for the stakeholders –i.e. the devices and the services providers.

Finally, Northrop et al. (2006) reveals the importance of the **autonomy** of large-scale system-of-systems. In fact, each of the sub-systems have to be able to run independently. If one of its dependency fails down, the system should be able to find an alternate sub-system to perform the same actions or nearly the same.

Controllability

While the system should be autonomous, it should not be to the detriment of the *controllability*. Administrators have to be able to manage and control the infrastructure's lifecycle.

A first criteria, is the financial **feasibility** (Firesmith, 2010; Weinstock and Goodenough, 2006): cost should be maintained into certain bounds. In fact, large scale systems usually require more resources. This often leads to increasing cost with less and less control on.

Therefore, Firesmith (2010) identifies the **testability** of the system as an important parameter. In fact, the use of tools such as simulators should be completed by testings as many real-world phenomena are too complex to be simulated.

Firesmith (2010) and Northrop et al. (2006) also take into account the **evolvability** parameter. The infrastructure should allow incremental evolutions of the M2M devices and services. These authors relate this property to the **observability** –i.e. being able to have a global vision of the system– and the **maintainability** –i.e. being able to maintain large number of geographically distributed resources.

1.3.3 Governance tactics for Scalability management

In order to deal with the scalability management, IT administrators can handle the infrastructure in several ways. In this section, we review the tactics set up in the literature to make IT systems scaling well. Such tactics define intermediate governance objectives and processes (governance step 2) in order to align the system with the requirements. But, in some cases, they might require a new governance cycle before being applied, then they drive the property adaptation needs (governance step 4). We can divide these approaches into three different tactics: (i) *scale-up*, (ii) *scale-out* and (iii) *scale-down*.

1.3.3.1 Scale-Up

For Michael et al. (2007), a *scale-up* consists in *"the deployment of applications on large shared-memory servers (large SMPs)"*. This definition can also be interpreted as *"adding resources to a single node in a system, such as adding memory or a faster drive to a computer"*[12]. In the context of the M2M, such a tactic could result in the use of more powerful servers to run the M2M platform. But it could also be an improvement of the gateways' hardware or even some M2M devices assuming some local coordination tasks (e.g. the repeaters).

But scaling-up a system is not necessary the result of an hardware improvement. Weinstock and Goodenough (2006) considers the *scale-up* approach as a process which *"handle[s] increased workloads without adding resources to [the] system"*. In fact, the scalability can be managed by algorithmic optimization. For example, Kessis et al. (2009) propose a middleware to adapt the system to the workload dynamically. Another example is the use of a cache (Neuman, 1994) which reduces the computation but requires more storage.

Further more, our review in Section 1.3.1 shows that a scale-up, along a given dimension, can be performed by accepting trade-offs along the other dimensions. For example, a possible trade-off is to tolerate a

[12]Definition used by the Wikipedia: `http://en.wikipedia.org/wiki/Scalable`

higher latency in order to maintain the *data loss rate* below an acceptable threshold. Further more, we can assume that even a “hardware scale-up” is the result of a threshold with respect to the *cost* of the infrastructure.

Nevertheless, the cost of such a scale-up approach is rather low (Michael et al., 2007). Another advantage is that it does not require any architectural changes. But it is very limited by the hardware and software capacity. For example, it is not possible to extend, infinitely and efficiently, the number of cores and their speed in a single server.

1.3.3.2 Scale-Out

The *scale-out* tactic consists of *“the deployment of applications on multiple small interconnected servers (cluster)”* (Michael et al., 2007). Instead of increasing, the capacity of a resource, the number of resources is increased.

For example, a service or a resource can be *replicated* on multiple nodes (Neuman, 1994). Thus, it is possible to access to it the using different servers. This reduces the load on each server and increases the availability and reliability of the resource. Usually, load balancing algorithms are used to manage the access the service and redirect the requests to the one of the servers which is less saturated (Boukhelef, 2010).

Another (complementary) approach is to distribute the resources along different servers (Neuman, 1994). Each server hosts one or several services. These services delegate sub-processes to others which frees capacity on their own server.

The main advantage of this approach is that the number of servers can be added dynamically to adapt the capacity to the workload. In fact, contrary to a scale-up, this approach has a potential of infinite capacity (Weinstock and Goodenough, 2006). But Bondi (2009) reveals that the role of the system architect is of first important in the case of scale-out systems. In fact, the cost of such a solution is more important.

Further more, it requires coordination mechanisms between the distributed services (Northrop et al., 2006). Thus, the complexity of the system is considerably increased which impacts the observability and maintainability of system.

1.3.3.3 Scale-Down

If the two previous approaches aim at managing an increased workload, the purpose of the *scale-down* tactic is to reduce the costs by adapting the system to a decreased workload (Fayad et al., 2000), though it is less considered in the literature. If the workload increases scale-up/out strategy is used. But when the workload decreases, the scale-down strategy is used to free the resources.

A scale-down strategy is implemented in the DASIMA middleware (Kessis et al., 2009) in order to switch between different configurations according to the scale of the system.

Further more, the cost of communication between the servers is not negligible and can reduce the performances. In fact, Northrop et al. (2006) note that efficient solutions for large-scale systems might be inefficient at lower scale. Hence, *on-demand* resource management is an interesting approach which is usually coupled with a “pay-per-use” model.

1.4 Synthesis

In this chapter, we have focused on the benefits and outcomes of M2M systems in order to point out the need for a clear governance. We have also studied how the IT governance could help us to design a governance for M2M systems. Finally, we took an example of an M2M governance issue: the scalability of the M2M infrastructure.

Based on classical IT and SOA governance principle, and considering the M2M specific issues, we identify the following five points to drive our work towards an M2M governance framework:

1. The requirements definition and modeling allows to align the infrastructure with the business requirements (see Section 1.1.3, governance steps 1–2).
2. Using a declarative approach (e.g. WSLA), such a definition can be done by non-technical staff while being understandable by the system.
3. In order to ensure the alignment of the governance objectives, the infrastructure must be monitored. To do so, it is necessary to place monitoring sensors into the governed system's infrastructure. In addition, the infrastructure should be adaptable in order to match the governance objectives (see Section 1.1.3, governance step 3).
4. Scalability management requires to be able to evaluate the performance based on monitoring and to compare them with the objectives. Based on these results, the governance should specify how the system or the objectives would be refined.
5. The governance objectives' refinement (see Section 1.1.3, governance step 4) is important to make the business objective match with the new infrastructure's constraints and opportunities.

In addition to these points, the opened, distributed, horizontal and large-scale nature of M2M system requires a flexible and reactive governance. Therefore, we believe the governance should be embedded in an automated loop.

The two next chapters will review the literature with these objectives in mind. We study how the M2M infrastructure can be adapted and controlled by another system in Chapter 2. Then we focus on Multi-Agent System (MAS) in order to express a governance, in 3.

For clarity purpose, we define the terms our governance analysis and proposition will refer to. Each of the following terms corresponds to a step in the governance cycle, which can be matched with the CobiT governance (see Section 1.1) and IBM's SOA governance (see Section 1.1.2) frameworks:

The Legacy System nominates the system to be governed. It consists of a set of infrastructure components (e.g. servers, gateways, wireless sensors. . .) and software modules used and/or usable to achieve the business objectives. Our governance system does not address the design of such a legacy system. One could use the governance definition to design it, however, this system can potentially be an already existing one.

The Governance Strategy refers to the global objectives of the system and the means to achieve them. It corresponds to the *IT Goals Definition* (CobiT) or to the *Governance Requirements Planning* (SOA) steps.

The Governance Tactic applies the Governance Strategy by the means of intermediate goals and processes. It corresponds to the *Process Goals Definition* (CobiT) or the *Governance Approach Definition* (SOA) steps.

The Governance Policy is the Governance enabler which monitors the system and applies the decided adaptations on its infrastructure. It corresponds to the *Activity Goals Definition* (CobiT) or the *Governance Model Enabling* (SOA) steps.

The Governance Refinement is the process through which the governance performance is measured and evaluated in order to improve and reorganize it. It corresponds to the *Governance Measurement and Realignment* (CobiT) or the *Governance Measurement and Refinement* (SOA) steps.

CHAPTER 2

Adaptation and Control of M2M infrastructure

Contents

As mentioned in Andrzejak et al. (2002), human intervention is a major cause of errors and system failures. Further more, the large scale of Machine-to-Machine systems makes such an intervention poorly reactive. This chapter aims at reviewing techniques used to adapt systems and define how relevant they can be to automate the governance of a Machine-to-Machine system. Therefore, it focuses on adaptive systems for Wireless Sensor and Actuator Networks (WSANs), network management and resource allocation.

Self-organizing systems are defined by *"the capability of adding and removing system parts without the need for reconfiguration nor the need for human intervention*" (Andrzejak et al., 2002). This is the subject of many forums in scientific community such as the *Self-Adaptive and Self-Organizing* systems conference series[1] (SASO), the *Software Engineering for Adaptive and Self-Managing Systems* symposium[2] (SEAMS) or the *International Conference on Autonomic Computing* series[3].

In this chapter, we consider the adaptation of a system as a control cycle which (1) monitors the state of the system and (2) tunes some of its parameters in order to make it work better. Hence, two types of control loop can be distinguished: *endogenous* and *exogenous*. The *endogenous* adaptation, studied in Section 2.1, consists of defining control mechanisms and adaption processes within the system itself. On the contrary, 2.2 studies the *exogenous* adaptation based on an external system specialized in the monitoring and the modification of the supervised system.

2.1 Endogenous Adaptation and Control

In endogenous self-adaptive techniques, adaptation is performed by the processes running the business logic itself. Supervision is done locally by the process itself which continually adapts and tune itself until it succeeds in performing its tasks.

Such techniques are used in several area in order to achieve tasks or solve problems which complexity is too big for deterministic approaches. This include efficient routing algorithms in large-scale networks (Schoonderwoerd et al., 1997) and WSANs (Erdene-Ochir et al., 2011; Iwanicki and van Steen, 2010) or resource allocation and the service placement in grids (Andrzejak et al., 2002; Kota et al., 2012), for examples. These systems need to be able to adapt to (i) the dynamics of the environment, the resources or the system itself, (ii) the distribution and (iii) the constraints of the resources. Further more, it is often the case no global information about resource availability and service demand is available in such systems, due to its distributed nature. In fact, sharing these information would lead to an overhead cost in term of communication, especially in large-scale systems.

This section reviews different techniques for designing self adaptive systems to identify the benefits and drawbacks. We first describe naive technique and then focus on more sophisticated techniques: swarm intelligence, clustering algorithms and Multi-Agent Systems (MASs). Finally, we synthesize the advantages and drawbacks of the endogenous approaches from a governance perspective.

[1]SASO conference series' website: `http://www.saso-conference.org/`
[2]SEAMS' website: `http://www.self-adaptive.org/`
[3]ICAC'13 website: `https://www.usenix.org/conference/icac13/`

2.1.1 Simple Adaptation: Probabilistic and Stochastic algorithms

Probabilistic and stochastic approaches are simple algorithms which requires no or few information. Therefore, they suit well to constrained environments. For example, Erdene-Ochir et al. (2011) propose a resilient routing algorithm for Low Power and Lossy Networks (LLNs). Given the high message loss rate in such networks, the messages are re-emitted redundantly, following a given probability, proportional to the distance to the sink node. This increase the fault-tolerance and the resilience of the network. But, it increases the load with potentially useless messages. Although, it requires no coordination messages.

Andrzejak et al. (2002) reviews two probabilistic approaches applied to the resource allocation problem in a grid: (i) the Greedy Algorithm and (ii) the Random/Round Robin (R3) algorithm.

The Greedy Algorithm consists of selecting the closest best solution. Therefore, it doesn't require much information or computation. For example, when a resource (e.g. a server in a grid) is overloaded, it pushes one or several tasks to the least loaded neighbor. This achieve load balancing easily. Nevertheless, the least-loaded servers receive tasks from all the others and quickly become overloaded. This might lead to oscillations and so, decrease the system performances. Moreover, it is not possible to guarantee the termination of such an algorithm nor it is to avoid cycles.

The Random Round Robin (R3) algorithm is typically designed for scheduling. But it is also suitable for many problems such as load distribution. This algorithm consists of the following steps: (i) when a resource is overloaded, an other one is chosen randomly, (ii) the latter either accepts the load or push it to another random resource, repeatedly until a resource is found, then (iii) the task migrates to the new resource. Such an algorithm is very simple and, above all, is stateless. However, cycle cannot be avoided and termination cannot be guaranteed, even if the latter issue can be passed over by limiting the number of hops.

These algorithms achieve good results with simple mechanisms. The main advantage of such an approach is that is requires no or limited overhead information and communication. In consequence, it provides good reactivity. However, it is hardly possible to guarantee the convergence of such algorithms. In addition, unexpected or unwanted behavior is often observed, which leads to non-optimal solutions or even worse performance.

2.1.2 Swarm Intelligence

Swarm Intelligence is inspired by the complexity of natural phenomena resulting from simple behaviors (Picard and Gleizes, 2004): ants, bees social spider, bird flocking or sand wavelets. . . The principles of swarm intelligence are *stigmergy* and *emergence*. *Stygmergy* is coordination mechanism using indirect communication such as the environment. The *emergence* is a collective behavior which cannot be described by the individual behaviors. Thus, simple programs are coded with a limited (or even no) reasonning cycle and coordinate together without direct communications. Yet the multiplication of such individual behaviors leads to the emergence of a more complex system.

Ant Colony Algorithm. One the most known example is the *foraging behavior of ants* Schoonderwoerd et al. (1997). In the nature, ants explore the environment randomly until they find food to bring back to the colony leaving a trail of pheromones. Then the other ants follow this track of pheromones to the food and on their way back leave pheromones too, enforcing the pheromone concentration. Longer paths will be less concentrated in pheromones which will evaporate while the shorter paths will be more and more concentrated. This phenomena leads to the emergence of a good path between the colony and the food. Further more, if a part of the path becomes unavailable, the ants will randomly try to get around the obstacle, creating a new good path.

Routing. Hence, this behavior have inspired the *ant colony optimization algorithm* to find good paths through graphs. If such an algorithm cannot guarantee to find the optimal path, it is very efficient in large graphs and very resilient. Therefore, the *ant colony optimization* algorithm have been used to solve several hard problems and especially routing and load balancing in networks. In fact, in Schoonderwoerd et al. (1997) the use of ants for the telecommunication network (based on the British Telecom network topology) gives significant improvement to the adaptation of to the number of calls and to the network topology.

Resource Allocation. Such an algorithm can also be used for the resource allocation problem. For example, Andrzejak et al. (2002) use a variation of the ant colony to manage the service placement in a grid network. Each server of the grid has a "pheromone table" which assigns for each couple service–server (s–S) a score depending on (i) the traffic cost between the server S and the servers hosting the dependencies of the service s and (ii) the server's load. These tables are updated by "ants" moving from one server to another in order to find a better placement for the service they represent. Such "ants" are generated by the services themselves when the score of a service's dependencies raises a critical value to find a better assignment (i.e. getting closer to the dependencies' servers and to a less loaded server).

Swarm intelligence approaches have proved their efficacy to deal with complex and distributed systems. Yet, the convergence of such algorithms is very hard to determine. It is not possible to define the expected results with respect to the governance objectives.

2.1.3 Clustering Algorithms

Broadcasting consists of sharing information (e.g. position, load, status. . .) with the entire system. This allows each part of the system to be up-to-date, and to take decision accordingly. But, such an approach scales badly as the resources quickly become overloaded (Andrzejak et al., 2002).

Clustering algorithms help to deal with the scalability issue by dividing the system into smaller ones. Communication is done within a cluster only and between cluster heads. In order to reduce the clusters' size, it is possible to build several levels of clusters, so called hierarchy. The smaller the cluster size is set, the higher is the number of levels. If a message must be send to a node outside the cluster, it is sent to the cluster head which tries to transmit the message at the upper level, and so forth.

Routing. Clustering appears to be an efficient approach for WSAN routing. Dressler (2006) shows that it enhances the performance of routing and it reduces the energy consumption. Several approaches are identified to achieve this goal. For example, passive clustering consists of setting the cluster structure in a fixed top-down manner. This reduces the overhead for the maintenance of the cluster structure, but lacks reactivity to changes. On-demand clustering is another approach that mitigates the need for permanent maintenance of clusters and the possibility to reorganize the cluster.

LEACH. Although, cluster heads, in hierarchies, can become bottlenecks if they are overloaded (Andrzejak et al., 2002). It is especially the case if the cluster heads are statically or externally defined. Heinzelman et al. (2000) have proposed the Low-Energy Adaptive Clustering Hierarchy (LEACH) to distribute the energy load in each node. Periodically, the cluster head is changed: each node chose to be a cluster head probabilistically for a round and broadcasts its decision, at the end of the round the others choose with an increased probability.

Gossiping. Another approach is gossiping: to avoid all-to-all communication each transmits the information received to its neighbors either deterministically or randomly. The PL-GOSSIP algorithm (Iwanicki and van Steen, 2010) maintains a recursive multi-hop area hierarchy in large WSANs. In order to ensure the quality of the network, each node considers other as neighbors if and only if it exists a high quality link in both directions. Then, the hierarchy and cluster heads are defined recursively, in a bottom-up manner. At level 0, groups are composed of individual nodes, which are their own cluster-head. For each level $i + 1$, the cluster head is the head of the central level i subgroup. Given a period T, each node broadcasts its topology knowledge which allow to take into account the changes in the network. In one hand, the redundancy of information increase the robustness of the network. And on the other hand, the value of the T parameter allows a trade-off between the energy consumption and the reactivity. In fact, the author demonstrate on a testbed that their algorithm consumes 33% to 55% of the energy consumed by classical clustering protocols for same reactivity. But, it is also from 2.6 to 3.1 times as reactive when consuming the same amount of energy.

As we can see, clustering enables to reduce the complexity of a system using a “divide and conquer” approach. This increase the scalability property of the system.

Nevertheless, by dividing it into several smaller systems, it raises the problem of (i) maintaining the structure of the clusters –i.e. how to build and update it– and (ii) synchronizing the different sub-systems. The former issue can be solved by different techniques such as gossiping, which reduces the data broadcast. The latter one is performed by the hierarchy of the cluster-heads. That is, cluster-heads have the responsibility to decide whether or not to transmit the information to the upper/lower level of hierarchy.

While it suits well when considering problems such as routing, it is more complicated when considering Quality of Service (QoS) and heterogeneous capabilities. This requires more “intelligence” in order to distribute the coordination between the sub-systems.

2.1.4 Multi-Agent Systems

Agent-based systems and MAS are specially adapted for dealing with the complexity and the dynamics of the systems (Jennings, 2001). Actually, agents are programs capable of reasoning about their environment in order to take the decision concerning the action to realize.

Mobile Agents. In Andrzejak et al. (2002) the Ant-Colony Optimization algorithm is adapted using mobile agents. Instead of simple ants maintaining tables of service assignment to servers (see Section 2.1.2), the agents evaluate the possible assignment of all the services to the current server and its neighbors. Further more, they do not die, but keep track of their evaluation table. This allows the agents to remember the previous good and bad assignments, and so, improve the optimization of the grid. But, the agents are heavier than simple ants. That is, it increases the cost the agent's computation and transmission on the network.

Embedded Agents Considering the aspects mentionned above, mobile agents would not suit to LLNs, but agents can be embedded into the sensor nodes. For example, (Carr et al., 2009) use embedded agents in order to deal with the energy trade-offs in constrained networks. Each sensor is an agent which sense a value, aggregates and transmits this value to a sink node. Each agents decide whether or not to transmit their aggregated value or not. This decision is made in order to balance the energy cost with other parameters, for example the global accuracy of the global aggregate value or the security –i.e. using a more or less secure/energy-consuming cryptographic algorithm.

Adaptive Multi-Agent System. Agents can be used for run-time software adaptation to deal with the dynamics of complex environment.

In particular, Capera et al. (2003) have proposed the Adaptive Multi-Agent Systems (AMAS) theory in which *"the MAS is able to change its behavior to react to the evolution of its environment"*. It is based on the concept of *emergence*. That is, each agent carries out a partial function leading to the realization of an emergent function by the AMAS. This approach suits to complex problems, where the global function cannot be expressed easily or would require to much computation to be solved. Hence, the agents self-organize in order to avoid so called *Non-Cooperative Situations* (NCS) based on their local perception of the system. When an agent detects such a situation, it interacts with the others in order to find a new (better) configuration or to bring the system back to a known good configuration.

Such an approach suits to design dynamic adaptive systems in which the agents have the freedom to adapt the system provided that they avoid the NCS. Such a design allows to focus on these objectives and new agents can be added to avoid new NCS if the governance strategy evolves. The main drawback of this approach is that convergence cannot be guaranteed as it is based on the principle of emergence. This bottom-up definition poses a problem in terms of governance as Information Technology (IT) boards usually use a top-down definition.

Multi-Agent Organization In some other approaches, the global objectives are defined through an explicit organization. As an example, Kota et al. (2012) use a MAS to manage an open dynamic grid computing-based Geographical Information System (GIS). Such a GIS requires an intense use of computing resources to perform its processes. More, openness means nodes in the grid may appear/disappear at run time, and the system' dynamics are related to the introduction of new tasks in the grid.

The MAS provides self-adaptation for the constrained distributed resource management, in which each agent represents a server in the grid accepting/rejecting tasks. The structure of the MAS is explicitly defined as a graph of relationships between the agents (i.e. acquaintance, peer, authority). Moreover, the tasks are modeled as a workflow of Service Instances, to be provided by the agents.

This explicit design allows the agents to reason about the actual organization responsively to the system's dynamics and openness. In fact, agents can decide whether they execute the task by themselves, or delegate it to a subordinate nodes, or try to delegate it to its peers.

One the one hand, such a design allows to define the objectives a priori which ease to make the system match the governance strategy. In this way, agents are aware of the objectives so they can self-adapt to meet them, according to their load and capabilities. On the other hand, the authors shows that such a reasoning and an adaptation have a cost which directly impacts the IT performance. In order to minimize this cost, they take into account the adaptation cost within a utility function leading to the decision whether to perform a change or not.

2.2 Exogenous Adaptation and Control

The previous section focused on the indogenous control of system. We revealed that such an approach lacks of separation of concerned and carries an important overhead. These two points lead to a poor capability for embedding a governance expression and reasonning. The goal of this section is to study adaptation of a system by an other system.

In this section, we consider systems which control other systems (*legacy system*). That is, while the *legacy* system runs the business logic only, the monitoring and the adaptation is performed by an other system. The latter monitors the *legacy system* using sensors on it and adapt it if need by (re)configure it. And so, such a systems can focus on the governance reasoning. Considering the exogenous control of IT systems, IBM (2006) identifies 5 levels of autonomicity:

1. **Basic:** managed by skilled IT staff,
2. **Managed:** monitoring tools to reduce the systems administration tasks,
3. **Predictive:** intelligent monitoring tools, recognize behavior patterns, suggest actions to IT staff,
4. **Adaptive:** able to take action to achieve the performance meeting the Service Level Agreement (SLA), defined by IT staff,
5. **Autonomic:** system/components managed by business rules and policies enabling the IT staff to focus on the business needs.

This section reviews the pros and cons of different techniques for designing such control systems: (i) middleware systems, (ii) autonomic computing, and (iii) agent- and MAS-based systems.

2.2.1 Middleware Approaches

The Wikipedia defines a *middleware* as *“a "software glue" [that] provides services to software applications beyond those available from the operating system. [...] It makes it easier for software developers to perform communication and input/output, so they can focus on the specific purpose of their application*[4]*.”* A middleware also *“enables the various components of a distributed system to communicate and manage data. It supports and simplifies complex distributed applications*[5]*.”*

Such a middleware layer provides an abstraction of the application’s physical deployment. Hence, the application runs the business logic while the middleware take care of the physical constraints, such as the distribution over the network or servers and network load.

In the following, we describe two examples of middlewares, the CLF middleware and the DASIMA middleware, and study how they contribute to the perspective of an M2M governance perspective.

2.2.1.1 The CLF middleware

Arregui et al. (2001) extend the Communication Language Facility (CLF[6]) to support scalability management. The CLF middleware aims at coordinating distributed active software components over a Wide Area Network (e.g. Internet). It is based on a high level declarative approach, which facilitate the transcription of the requirement objectives to the middleware. The CLF middleware can be divided into two parts: the *object model* and the *protocol*.

The object model consists of enriching the applications’ objects so they are viewed as resource managers. On the one hand, the CLF interface allows to invoke the resource’s regular methods. Hence, the access to the object is hidden by the CLF manager, which can possibly deploy several instances of the same resource on different hosts. On the other hand, it provides a clear separation with the management concerns through the CLF services. To do so, it adds new primitives to interact with the object:

- inquire and negotiate objects capabilities in term of resource availability,
- perform basic transaction operations over the resources of several objects (i.e. two-phase commit),
- request new resources to be inserted.

The protocol is based on the Coordination Scripting Facility (see Listing 2.1, p. 50). It provides a *high level declarative specification* of coordination of CLF services (e.g. insertion, removal...). Such a script bounds service interfaces to the legacy objects which provides a representation of the application (Line 1). In a second section (Line 10), it defines a set of rules defining how those services should be processed.

At run time, these scripts are enacted by coordinators which manage the resources (e.g. instantiation, deletion...). A coordinator tries to find the resources corresponding to the tokens in a rule and fully instantiates it with for each set of consistent values satisfying the rule’s constraints. Then, it processes the resources

[4]*Middleware*, in the Wikipedia: `http://fr.wikipedia.org/wiki/Middleware`

[5]*Middleware (distributed applications)*, in the Wikipedia: `http://fr.wikipedia.org/wiki/Middleware_(distributed_applications)`

[6]The Xerox European Research Center CLF webpage: `http://www.xrce.xerox.com/Research-Development/Historical-projects/CLF-The-Coordination-Language-Facility`

following the right side of the rules (delimited by <>-). In addition, it is possible to generate script rules "on-the-fly" which allows for dynamic resource management.

High Level Declaration. The CLF rules are used to define how the middleware system should handle the scalability concerns. This is done by specifying high level rules to be handled by the middleware. These rules are independant from the deployment and offer much flexibility.

For example, in order to reduce the network load, the rule Line 11 ensures the *postscript printer* service selected is host on the same site as the file to treat. The graceful degradation policy implementation (Line 20) successively select lower quality services in order to ensure the *summarize* service is delivered within a limited time.

Finally, an example of load balancing is given Line 31 which migrates user data for the *ProfileDirectory* service when the amount of users on the same server raise a certain threshold.

To conclude, we can observe interesting properties for governance enactment that the CLF middleware has:

1. separation of concerned is realized through the encapsulation of the legacy system and the enrichment of the objects with management interface,
2. the high level declarative specification of the coordination mechanisms facilitate the transcription of the governance policy to the software coordinators using first order logic rules,
3. the software coordinators enact the governance policy by evaluating the rules and processing the actions using the services' CLF management interface.

Such a middleware suits well to the scalability management of a dynamic distributed system. Nevertheless, we can notice that the rules are very specific to the resources' business process. That is, the functional and non-functional requirements are not clearly identified. What's more, the concepts of QoS and SLA are implicitly defined, and so, it is hardly possible to automate the reasoning about the governance policy.

2.2.1.2 The DASIMA middleware

Kessis et al. (2009) apply the DASIMA management middleware to the management of the software layer responsible for M2M data processing. That is, managing hundreds to thousands of machines that are heterogeneous and geographically distributed. The DASIMA middleware's architecture use the Fractal component-based framework and is divided into three layers.

The managed resource layer is the lowest layer. It is a representation of the real world managed resources. Such resources can be logical (software) or physical (hardware). This layer allows the middleware to be aware of the system's state and to act upon it.

Each resource is represented by a so-called *Managed Element* which provides information about the managed resource. But it is also a wrapper which enables an homogeneous management of the heterogeneous resources (e.g. logical/physical).

```
interfaces:
  docStatus(inforsourceId,docId,status):
    -> infosourceId,docId,status
    is LOOKUP ConfigurationManager.Status
  accessRight(infosourceId,docId,right):
    infosourceId,docId,right -->
    is LOOKUP ConfigurationManager.AccessRight
[...]

rules:
/* Locality rule */
  printReq(infoSourceId,docId,printerId) @
  `site(infoSourceId,site1) @
  `site(printerId,site2) @ sameSite(site1,site2) @
  `localYellowPages('toPs'.pbj,srv) @
  `content(infoSourceId,'Content',docId,content) @
  toPs(obj,srv,content,postscript)
  <>- printer(printerId,'Print',postscript)
[...]
/* Graceful Degradation rules */
  summTicket (infoSourceId,docId,obj,srv) @
  `content (infoSourceId,'Content',docId, content) @
  summarize(obj,srv,content,summary)
  <>- storeSummary(infoSourceId,docId,summary)
  /* Degraded service selection */
  summTicket (infoSourceId,docId,obj,srv) @
  wait('600') @
  `summPolicy(obj,srv,nextObj,nextSrv)
  <>- summTicket(infoSourceId,docId,nextObj,nextSrv)
[...]
/* Dynamic Load Balancing rules */
  `yellowPages('ProfileDirectory',src) @
  `site(src,site) @
  `nb(src,nb) @
  greaterThan(nb,'100') @
  generateName (src,dest)
  <>-
  launch(site, 'ProfileDirectory',dest) @
  migrationTicket(src,dest)
  /* Migrate data */
  `migrationTicket(src,dest) @
  `profileDirectory(src,'nb',nb) @
  greaterThan(nb,'50') @
  profileDirectoryData(src,'UserData',data)
  <>- profileDirectoryData(dest,'UserData',data)
  /* Stop migration */
  migrationTicket (src,dst) @
  `profileDirectory(src,'nb',nb) @
  lessThan(nb, '50' )
  <>-

end
```

Listing 2.1: Example of a CLF script

The resources are grouped into *management domains* corresponding to an abstract representation of, e.g. , applicative groups, firmware update groups, radio management groups…

The middleware layer is the core of DASIMA. It offers management mediation functions to the upper level. Such functions allow the system to be supervised and controlled by management system administrators.

These management components can be plugged during run-time, and so adapt to the dynamics of the managed system. In particular, they provide support for different management topology of the management domains. These domains can be managed by a single manager, duplicated managers for resilience purpose, specialized managers, or hierarchical manager to improve their locality. Such a choice is made case-by-case and is left to the management administrators.

Further more, using the resource wrapping, at the lower layer, they can adapt the system on two levels: (i) at the architectural level, by adding/removing instances of components and services, and (ii) at the behavioral level, by changing the execution of the components and services (e.g. configuration of polling frequency, filters…).

The management application layer is the top layer. It is composed of a set of applications that contain the management logic as expressed by the administrators. Such management applications can be user interfaces for the administrator to configure the management middleware. But it can also be software controllers which changes this configuration when management policy is not suitable.

Thanks to this architecture the DASIMA middleware addresses three challenges related to the M2M data processing. First, the lower layer abstraction allows to take into account of the heterogeneity of the resources while providing a uniform interface for managing them. Secondly, the volatility of the resources and topology by dynamically (un-)plugging management components. Finally, multi-scale management is provided by switching from a topology (e.g. centralized) to another (e.g. hierarchical) depending on the system's scale. Moreover, DASIMA is a reflexive middleware, that is, it is able to manage itself using management components which themselves control the other management components.

Middlewares seems to offer a good paradigm for controlling and adapting distributed systems. Thus, it suits to the enactment of the governance policy within the system.

Nonetheless, the underlying adaptation concerns and reasoning are not explicitly defined in the middleware model. For example, the DASIMA middleware offers the possibility to define the governance whether through an administrator user interface or via a software agent capable of reasoning about the middleware layer and to reconfigure it. But, the governance and adaptation models are defined externally from the middleware itself.

2.2.2 Autonomic Computing

The *Autonomic Computing* paradigm aims at fulfilling the gap identified in the previous section, that is, an end-to-end framework for an autonomic management of the IT infrastructure. It usually refers to four categories (so-called "disciplines") of autonomic components: (i) self-configuring, (ii) self-healing, (iii) selfoptimizing and (iv) self-protecting systems (IBM, 2006). A survey on Autonomic Computing (Huebscher and McCann, 2008) reveals it is applied to many fields, including grid computing, ubiquitous services and WSAN.

2.2.2.1 Autonomic Computing Architectures

Although several work addressed this domain before, the main contribution to the Autonomic Computing was brought by IBM. IBM (2006) proposed a reference architecture for designing an Autonomic Systems. This reference model offers a clear separation of concerns and a unified process for adaption the IT. Figure 2.1 and Figure 2.2 give an overview of such an architecture.

At the base of this architecture is the Autonomic Element. It is responsible for monitoring and managing a part (e.g. a server, a component or a service) or a property of the system (e.g. security or performance). An autonomic element implements the Monitor–Analyze–Plan–Execute–Knowledge (MAPE-K) control loop, which is separated into five modules, as given in Figure 2.1), corresponding to the different concerns:

1. the **monitor** module diagnoses the *symptom* of the system, based on sensor events;
2. the **analyze** module analyzes the *symptom* and proposes a *change request*;
3. the **plan** module plans the changes to perform reported as a *change plan*;
4. the **execute** module performs the actions defined in the *change plan* through the effector interface;
5. the **knowledge** is a base shared by the four other modules to store the *facts* and the *decisions*.

Such an Autonomic Element takes place within a five layer reference architecture managing the overall system. As shown in Figure 2.2, the five layers are (from bottom to top):

1. **Managed resources:** they are legacy hardware/software components of the managed system. These components might embed their own intelligent control-loop –i.e. implementing endogenous algorithm as in Section 2.1– which can be visible or not to the management system.

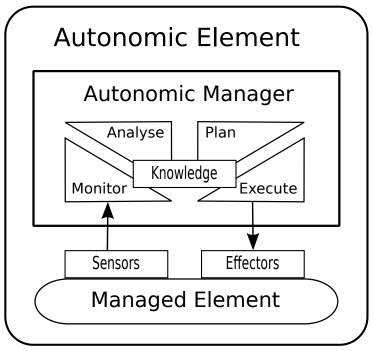


Figure 2.1: The Monitor–Analyze–Plan–Execute–Knowledge (MAPE-K) Control Loop of an Autonomic Element.

2. **Touchpoints:** they implement sensor and actuator behavior over the managed system and offer manageability mechanisms and interfaces. For example, if the endogenous adaptation algorithms are visible to the autonomic managers, they can (re)configure them through the *touchpoints*.

3. **Touchpoint autonomic managers:** they manage one or several resources using the touchpoints and embody different intelligent control loops. Most autonomic managers use goals or objectives to govern the behavior of the control loop.

4. **Orchestrating autonomic managers:** they provide coordination to deliver a system wide autonomic computing behavior.

5. **Manual Managers:** they provide a common system management interface for the IT staff.

Such a hierarchical architecture enables to control the autonomic controllers by other autonomic controllers, that is, using a uniform design paradigm. Each level of controllers considers a higher level of abstraction. Hence, touchpoint autonomic managers apply concrete policies to the management of the system, while orchestrating autonomic managers implement the higher level policy defined by the IT staff. For example, the latter governs the *self-configuration* policy of the whole system, while the former governs it for each component or sub-systems. It is also possible for the orchestrating level to be in charge of governing

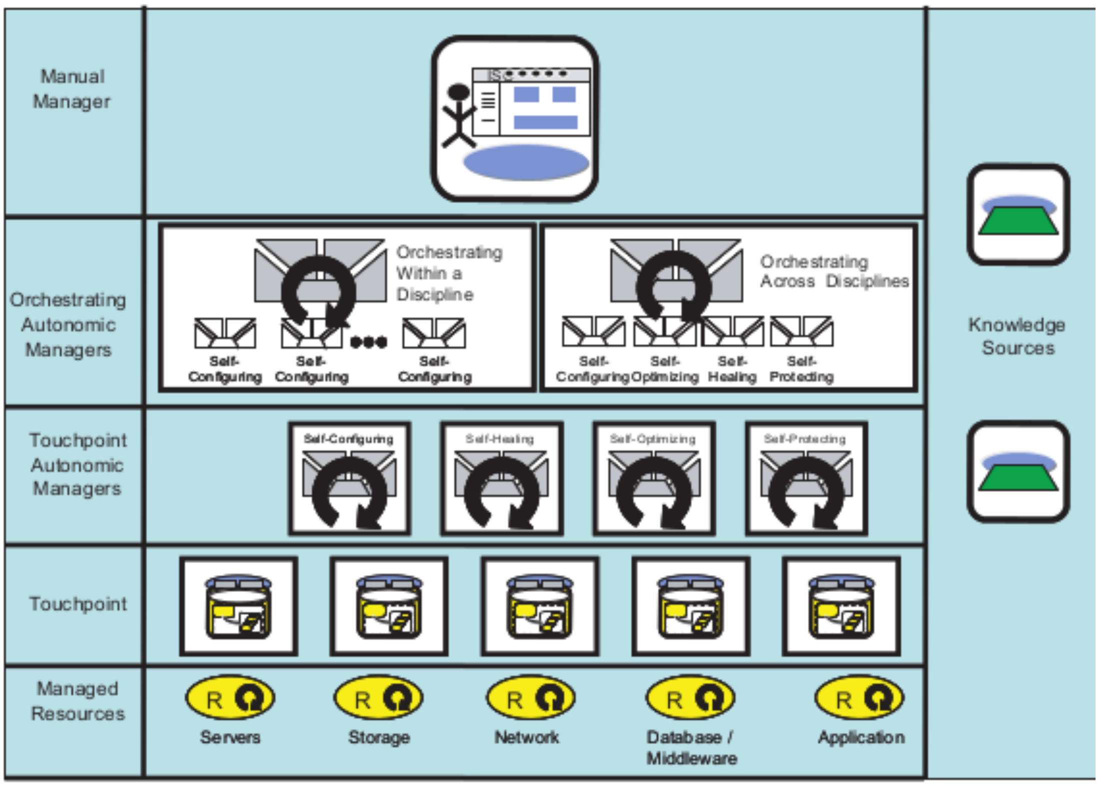


Figure 2.2: The Autonomic Computing five-layer architecture.

several "disciplines" for a component by managing specialized managers.
Nevertheless, such a hierarchy also suffers from its centralization, which can lead to bottlenecks when considering very large-scale systems. Not only the controllers, but also the decentralization and coordination concerns requires a particular attention from the designers.

2.2.2.2 A Decentralized Model for Autonomic managers

Weyns et al. (2010) propose a reference model for decentralized autonomic managers. This model defines a set of capabilities required for engineering decentralized self-adaptive system.Figure 2.3 provides a graphical representation of such a reference model. It is composed of one or several, potentially local, *Self-Adaptive Units*. A Self-Adaptive Unit adapts a local system based on *Meta-Level Computations* and *Meta-Level Models*.

Meta-Level Models Weyns et al. (2010) identify four types of *Meta-Level Models*:

the *System Model* represents the managed system, or parts of it;

the *Concern Model* defines the objectives of the Self-Adaptive Units, that is an optimization problem or a constraint satisfaction problem;

the *Working Model* contains the data structures and the information shared between the Meta-Level Computation modules;

the *Coordination Model* is the set of data required by a Meta-Level Computation module to coordinate with those of other Self-Adaptive Units.

Meta-Level Computations are components that reason about the Meta-Level Models and acts upon them. The Meta-Level Computations components follow the Monitor-Analyze-Plan-Execute (MAPE) model as defined previously. Coordination mechanisms are based on the Coordination Model, so that computation concerns can be independent from this model. Hence, different coordination mechanisms can be chosen depending on the coordination needs of the different Meta Level Computations.

For example, (i) the *Monitor* modules will use gossiping or local broadcasting algorithms, to update the System Model based on the observed data, and (ii) the *Analyze* modules will use auctioning or voting mechanisms, to analyze possible reconfigurations and trigger the suitable plan computation, while (iii) the *Plan* modules might require a synchronized coordination to plan the improved deployment, and then (iv) the *Execute* modules can performed the changes on the managed system independently from the others.

Such a decentralized model allows the designers not only to define the autonomic managers but also the mechanisms that suits best for the controllers' coordination depending on the specific need of the processes.

Still, the authors identifies many issues resulting from the decentralization. First, multiple *coordination overheads* can be observed as it uses more resources and it impacts the time for deciding and planning the adaptation. Then, as each local unit has a partial knowledge, *the environment model may become inconsistent* with the actual managed system, and can lead to sub-optimal solutions.

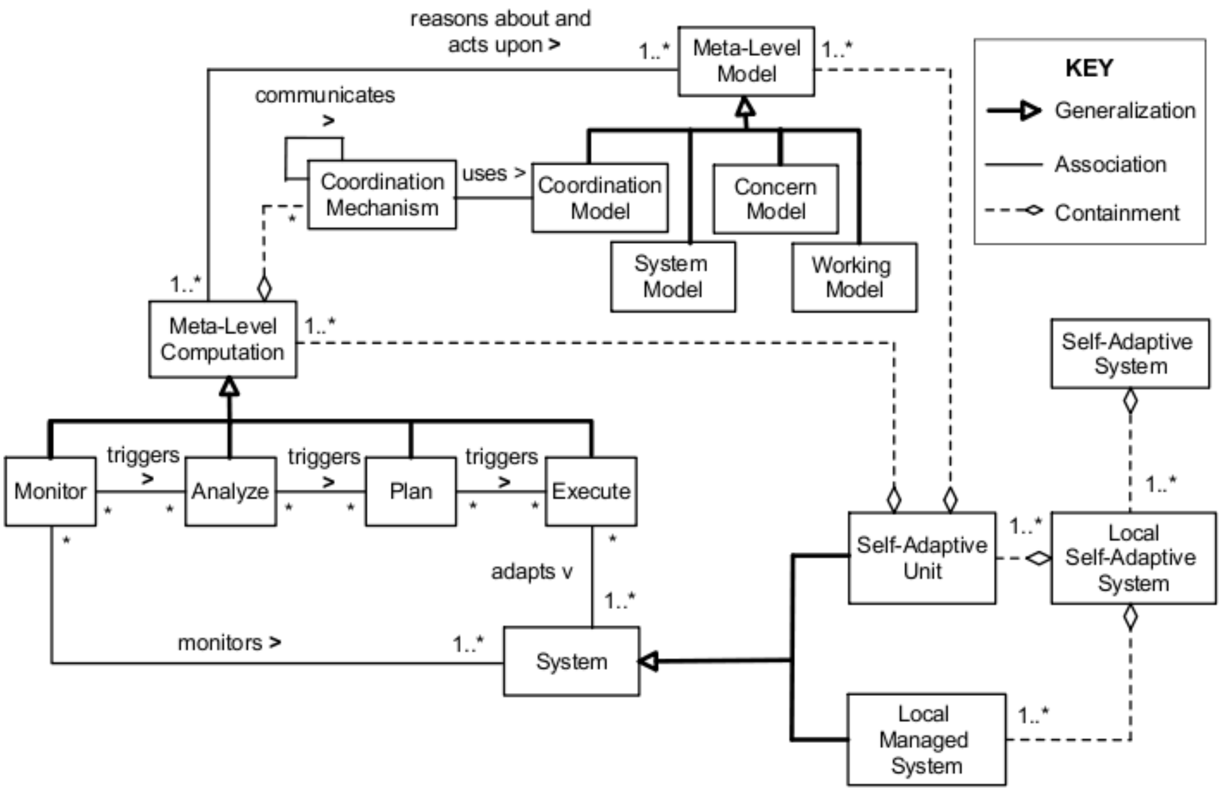


Figure 2.3: Reference model for decentralized self-adaptive systems (Weyns et al., 2010).

In addition, the local units might have *conflicting goals*. That is, conflicts between each individuals' objectives and a system-wide solution. The authors recommends to deal this problem with market based or negotiation mechanisms, eventually coupled with reinforcement learning. But, this will lead to more overhead and, more, it offers *no guarantee to converge to a stable system*.

Finally, it is hardly possible to provide a systematic engineering process, as many trade-offs must be defined in the design decisions. For example, different kind of organization design (e.g. peer-to-peer, hierarchical...) can be used, but it must be defined according the actual needs of the governed system.

2.2.2.3 Reasoning about the Autonomic Manager

Maurel (2010) observes that most autonomic managers are monolithic or lack for separation of concerns. Therefore, it proposes a framework for extensible and dynamic Autonomic Managers (CEYLAN). It is based on an hybrid component-based and WS-based approach to provide more flexibility to the autonomic managers.

Such an autonomic manager is decoupled in 3 layers:

1. the **Task Manager** which is an asynchronous and independent entity responsible for a single task in the control loop (i.e. data acquisition, control, planning...),

2. the **Task Controller** which defines the tasks for managing the tasks' life-cycle, communication, conflicts and persistence of events,
3. the **Administration** layer, which is responsible for the management of the autonomic manager. This layer contains two part: (i) the management policy that contains a description of the deployed tasks, their configuration and layout, and (ii) a description of the configuration of the controller tasks, done via a User Interface for the Administrators and/or the Autonomic Experts, or via Software Policy Manager.

These three layers enable to decouple the different concerns by bringing several layers of abstraction and reasoning. In particular, Maurel (2010) considers further evolution of the Administration layer. Among them, it identifies the need for a policy adaptation mechanism to check the coherence of the autonomic manager's behavior and policy, and to reason about coordination between the several autonomic systems.

This point of view is also supported by Huebscher and McCann (2008), which argue for a further level of interest: *Closed-Loop Autonomic Computing*. That is, systems that both are autonomic and allow the intelligence to drive the self-management to grow and refine itself. It consists of changing the way the logic of what to do should be defined. Instead of defining this logic statically and only update it by the parameters being fed into the loop, closed-loop autonomic should evolve the logic that drives the system depending on how well the "old" logic did.

According to Huebscher and McCann (2008) and Maurel (2010); Weyns et al. (2010), Multi-Agent Systems are a key technology to support such a governance reasoning.

2.2.3 Autonomous Agents and Multi-Agent Systems

Autonomous Agents and Multi-Agent Systems (MASs) have proved their capabilities to provide adaptivity and reflexivity properties to governed systems for many years now. In this section, we study how such technologies are used in the literature for managing, controlling and adapting exogenous systems, in particular M2M systems. Five example of such use of MAS are given in order to illustrate the relevancy of such technology. A deeper study of MAS frameworks and adaptation mechanisms is given in Chapter 3.

2.2.3.1 Multi-Agent Systems and Middlewares

MAS can be used together with a middleware in order to abstract the environment and separate the concerns between the business entities and the adaptation logic. For example, the AgentScape[7] (Wijngaards et al., 2002) middleware is used in Overeinder et al. (2002) to support grid networking. It is based on two entities:

Agents that are proactive entities, which interact with each other and invoke the resource objects,

Resource Objects which only perform computation and are reactive to the agents' initiative.

The middleware is a location manager for the grid's resources which hosts the resources objects and the agents. It hides all the details about the OS and implementation languages and provides and performs basic functions (e.g. asynchronous communication, tuple space, agent migration). Thus, it enables to abstract the

[7]AgentScape – Distributed Agent Middleware, available at: `http://www.agentscape.org/`

physical topology of the grid as all the agents' interactions and calls are exclusively done via the location manager.

Thanks to such a middleware, the agents can focus on the choice of the location to execute the computations, based on the information provided by the Location Manager (e.g. available resources, performance statistics).

2.2.3.2 Multi-Agent Systems for the Internet of Things

Kousaridas et al. (2010) uses a MAS for network monitoring in the context of Future Internet. Each device of the network –e.g. routers, gateways, access-points, sensors...– are controlled by so-called Cognitive Network Managers (CNM) and Domain CNM.

Cognitive Network Managers are agents implementing a Monitor-Decide-Execute decision cycle. In order to maintain good scalability performances they avoid a global vision of the overall network. Instead, they take local fast reactive decisions based on a situated knowledge.

Domain CNM are more sophisticated CNMs. That is, they perform the same decision process, but consider a higher-level knowledge in order to take global decisions. As a consequence, their decisions' response time can be longer than CNM's ones.

Using these two levels of decision taking, it is possible to enact fast adaptation policy –at the lower-level–, while reasoning about the adaptation policies –at the upper-level– in the mean time. Nevertheless, such an approach is limited in terms of expressivity of governance policies and strategy. In fact, it implies the Domain CNM agents know the adaptation policies the CNM agents can enact and implement a specific strategy the fine-tune them. But they cannot reason about such a strategy.

Oyenan et al. (2010) uses an organizational specification to separate the Control and the Execution internal processes to manage the energy consumption in WSAN. It uses two kind of entities.

mote agents that are embedded in the sensor motes and perform reactive operations. The Control process receives the agent's assignments and forwards them to the Execution process. The Execution process triggers the sensor mote's network and business operations according to the assignments.

base station agents are external agents that are responsible for the organizational decisions. They assign the mote agents their role in the network in order to save their life-span and to achieve the organizational goals. The Control process uses a a simple greedy algorithm for the role assignment, and the Execution process assigns the motes their roles. They achieve self-configuration and self-healing of the WSAN by distributing the energy consumption uniformly over the motes.

Using an organizational definition, the agents are able to reason about the system's goals and its functioning and constraints. They can take decisions accordingly and, potentially, reason about the adequacy of the global objectives. As stated in Section 1.4, such a reasoning is necessary to take into account the M2M infrastructure's dynamics in the governance process. It enables to consider the openness and the multiple stakeholders' requirements regarding the actual system's functioning and constraints.

2.2.3.3 Multi-Agent Systems for Dynamic Open Systems

To deal with open and dynamic systems, Aldewereld et al. (2010) propose the ALIVE framework, based on the AgentScape middleware. It aims to support the design, the deployment and the maintenance of distributed systems of services. Such a support is performed at three levels:

1. A Multi-Agent Organization –OperA[8] (Dignum, 2004)– is used to allow the coordination, the reorganization and the adaptation of Web Services. Such an organization represents the organizational stakeholders and their relationships. The organizational goals, requirements and restrictions provide the agents the context for coordination and the services to provide. Using norms to regulate the organization brings more flexibility. Further more, the organization verifies the system's compliance with organizational specifications and handles the changes to the organizational specifications.

2. The agents follow the operational constraints defined in the organization level. Coordination Agents enact the workflow tasks corresponding to the plan selected by the Planning Agent, and so, they transform the organizational specifications to a coordination plan, representing an interaction pattern. The enacting of the workflow task is performed by making use of the service level.

3. The service level is used to enable the adaption to the dynamic nature of the Web Services at runtime. To do so, the AgentScape middleware embeds (i) a semantic description of the services, (ii) the selection of the most appropriate service for a task, and (iii) monitoring of the service activities.

Such a Multi-Agent oriented approach brings new tools and methodologies for adaptive systems. In particular, it provides a higher level of abstraction allowing to model different stakeholders and their requirements. In addition, agents exhibit good capabilities regarding coordination, adaptation and organizational reasoning.

So, it makes MAS an interesting technology to make a system more than autonomic, but also capable implement a full governance cycle.

2.3 Synthesis on control and adaptation of systems

After reviewing adaptation mechanisms in the two previous sections, we now synthesize and compare them with respect to the perspective of building a governance system.

- On the one-hand, *endogenous* self-adaptive systems embed adaptive algorithms directly within the system's components.

  We can note many advantages considering theses systems: (i) fast reactivity to the environmental dynamics and to failures due to local adaptation behavior, (ii) no/little central management is necessary to keep the system operational as it is not necessary to maintain a global state, representation, thus it reduces the management overhead, (iii) this decentralization also provides good scalability properties.

[8]The OperA framework's website: `http://www.cs.uu.nl/research/projects/opera/`

Nonetheless, such an approach suffer several drawbacks identified by the authors we cited: (i) lack of controllability as the predictability of the behavior decrease while the scale increase, (ii) such complexity raises software development issue – where to run which part of the system? how to distribute the data? how to test the system? –(iii) the interference of such multiple cross-mechanisms can lead to unforeseen effects (i.e. "negative emergence"), and (iv) the global accuracy of the system may suffer from too much adaptation as reasoning about the system and modifying its structure and objectives have a non-negligible cost. From a governance perspective, the main issue is the endogenous approach mixes the business logic and the adaptation logic. It results in a non-negligible overhead to be supported by the infrastructure. In addition, it make the governance concerns less expressive and spread over the governance processes at run-time.

- On the other hand, an exogenous control system allows for a clearer separation of concerns and a higher abstraction level. For example, the autonomic computing architecture, reviewed in Section 2.2.2.1, is based on a layered abstraction model, making the system's control closer to the business concerns than the implementation concerns.

  In addition, the governance overhead cost can be externalized (i.e. distributed across external platforms) more easily. While such an externalization of the adaptation is less reactive, due to the latency of the sensors and effectors, the governance system can manipulate endogenous adaptive rules making the legacy system more-or-less reactive and adaptive. In the mean time, it can monitor and control the efficacy of the embedded algorithms and tune them or replace them.

  As a consequence, an exogenous approach seems more appropriated to a governance system.

- We have also pointed out the interest of MAS technologies as a promising paradigm for implementing the different governance concerns.

  In fact, MAS are decentralized softwares capable of coordination and adaptation. Agents are decision making programs that allow to automatically reason about both the system –like autonomic manager programs– and the governance policies. MAS have been used in many fields of computer engineering such as middlewares, Service Oriented Architecture (SOA), Internet of Things (IoT) where their ability for managing open, dynamic and constrained systems.

  In the next chapter, we focus our review on MAS from a governance perspective.

CHAPTER 3

Multi-Agent Systems from a Governance perspective

Contents

The techniques reviewed in the previous chapter reveal the relevance of Multi-Agent-based approaches for building distributed adaptive systems.As stated in Brazier et al. (2010), governance seems to be a "natural function" for Multi-Agent Systems (MASs). In fact, agent technologies focus on collective decision making within distributed environment. Though, this assertion rises several questions: "how to explicit the governance strategy in MAS?", "how agents can reason on these objectives to make the system comply with the strategy ?" and "how to monitor and control the governed system?"

The goal of this chapter is to review adaptation processes in a MAS usable for a Governance system. First, it analyzes the Multi-Agent Oriented Programming approach from a Governance perspective in Section 3.1. Then, Section 3.2 compares the different approaches for expressing policy and collective behavior in order to control and adapt a legacy system. Finally, Section 3.3 compares the different approaches for reorganizing a MAS.

3.1 Multi-Agent Oriented Programming

Agent-based and Multi-Agent System (MAS) technologies are recognized to be very suitable to deal with the complexity of large scale systems, dynamic environments and autonomous software (Jennings, 2001). Nevertheless, such complex systems require a difficult conception and development phase. Therefore, it is important to identify and separate the different concerns of such system from both the conceptual level and the implementation level (Melliar-Smith et al., 1997).

The Multi-Agent Oriented Programming (MAOP) approach is based on a synergistic combination of the four dimensions of MAS programming concepts (Boissier et al., 2011). These four dimensions are also known as the VOWELS dimensions (Demazeau, 1995): 1. the Agent dimension, 2. the Environment dimension, 3. the Interaction dimension and 4. the Organization dimension (see Figure 3.1). Each of these dimensions addresses a different concern of the MAS. Hence, MAOP helps to reduce the complexity and to localize the intelligence in the MAS (Jennings, 2001). Further more, different tools can be used and combined to model and develop these different dimensions.

3.1.1 The Agent Dimension

The Agent dimension is the active part of the MAS. It is composed of several agents wich involve reactive and proactive decision processes. Each agent focuses on a part of the system only. A collective behavior is expressed by the interactions between the agents with each others and with the environment.

Agent Oriented Software Engineering (AOSE) tackles the problem of designing and implementing agent software. The Agent Unified Modelisation Language (AUML) (Bauer et al., 2001) provides a tool for designing the internal behavior of an agent, based on the object-oriented UML approach. In particular, it makes possible to define entities using the Belief–Desire–Intention (BDI) paradigm (Bratman et al., 1998).

The BDI Agent Model is inspired by the human reasonning cycle and decomposed as following: *beliefs* are the information the agent knows from its own perceptions or from others' statements (e.g. the state of the environment it can perceive, the capacities of other agents. . .), *desires* are possible (internal and/or external) states the agent might want to reach which will guide the agent's decisions, and *intentions* can be considered

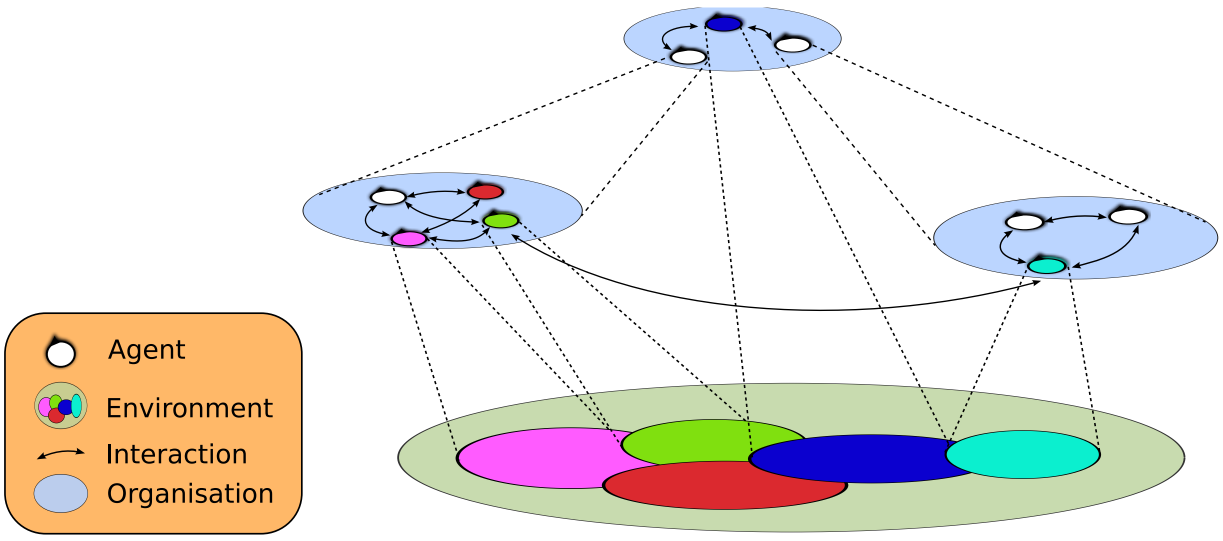


Figure 3.1: View of a Multi-Agent System and its four dimensions: Agents, Environment, Interactions and Organizations. (Adapted from Jennings (2001))

as goals selected by the agent, based on its beliefs and desires, and structured by plans which decompose the intention into sub-goals until they can be achieved directly.

BDI has become the main reference architecture for agents, and subject to many implementation languages. Busetta et al. (2000) have proposed the JACK Intelligent Agents framework which provides components and tools for building BDI agents. Another implementation of such a BDI model is the *Jason* language (Bordini et al., 2008). It based on the first-order logic programming language AgentSpeak(L) Rao (1996), and provides a Java interpreter for this BDI language.

3.1.2 The Environment Dimension

The Environment dimension is a domain dependant first-class abstraction in which the agents are situated. The environment represents resources which can be shared between the agents. Agents use sensors and actuators to monitor and act on the environment. The environment can be used as a communication channel for the agent, like in stigmergic approaches where agents do not communicate directly with each other.

Therefore, the environment is a suitable abstraction for representing the external (to the MAS) applications the MAS interacts with (Boissier et al., 2011; Brazier et al., 2010; Dragone et al., 2011).

Weyns et al. (2007) define the *environment* as *"a first-class abstraction that provides the surrounding conditions for agents to exist and that mediates both the interaction among agents and the access to resources"*. This definition states that the environment should be an explicit part of the MAS providing to the agents, situated in this environment, an abstraction level of the non-agent resources but also a design space containing the communication, coordination and mediation mechanisms.

Figure 3.2 illustrates the reference model proposed by Weyns et al. (2007). It contains a *State* module that maintains the actual state –i.e. the context– of the MAS environment. The *Translation* module allows the

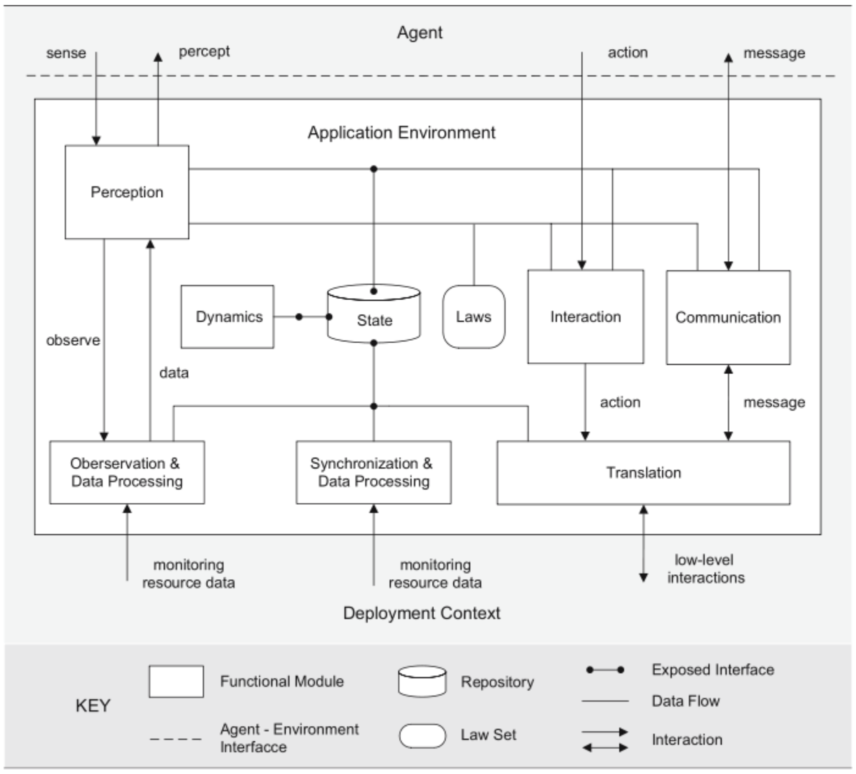


Figure 3.2: A reference model for the environment in a MAS. The environment abstracts the external resources and facilitate agents' interactions and regulation. (Weyns et al., 2007)

environment to interact with external systems. Together with it, the *Synchronization & Data Processing* and the *Observation & Data Processing* modules feed the state with data from the monitored external resources, while the *Dynamics* module contains rules to make this state evolve. The *Perception* module is an interface for the agents to be aware of the agents and resources as well as the events happening in the environment. Based on this perception, the agents can decide to initiate interactions and or send messages. These are treated by, respectively, the *Interaction* and the *Communication* modules, which handle them and address them to the appropriate recipient –e.g. broadcast, an other agent, a resource. Finally, the *Laws* module enacts the organizational specification and regulates the agents' perception and interactions as well as the access to the resources. Hence, the environment is a first-class abstraction that links together all the other dimensions of the MAS.

Omicini et al. (2008) give a conceptual foundation, called the Agents & Artifacts (A&A) meta-model, to implement such an environment model. This meta-model defines three levels of abstraction: (i) the *agents* corresponding to the pro-active processes of the MAS, (ii) the *artifacts* encapsulating the execution of autonomous passive activities of the MAS (e.g. resources, communication media) shared and manipulated intentionally by the agents, and (iii) the *workspaces* as logical containers in which the artifacts and the agents are situated. This approach is implemented by the CArtAgO framework (see Section 3.2.5 for more details).

3.1.3 The Interaction Dimension

Interaction is a particular dimension of the MAS: agents interact with the environment to act and sens it, with other agents and with the organization, but the objects and artifacts of the environment also interact with each other and so do the different parts of the organization. Therefore, interaction can be consider as a transverse dimension along the others.

The Interaction dimension in MAS is principally based on the *speech-act* theory, that addresses interactions between the agents. Most of the effort in this part have been done by the Foundation for Intelligent Physical Agents (FIPA)[1], which released the FIPA Agent Communication Language (FIPA-ACL) standard protocol, and the Knowledge Sharing Effort (KSE)[2], which released the Knowledge Query and Manipulation Language (KQML) (Finin et al., 1994).

The *speech-act* theory considers that a speech is an action of the speaker on the environment with a purpose (e.g. ask for a resource, give an order...). To understand each other, agents share a common ontology to communicate.

A popular implementation of FIPA-ACL is the JADE middleware[3] (Bellifemine et al., 2008) which provides direct ACL-compliant communications between the agents. Nevertheless, it has no mechanisms for managing ontology mediation. Thus, the agents must share implicitly the same ontology.

Other approaches try to provide a more complete framework for designing the interactions, for example IODA (Kubera et al., 2008) and MERCURIO (Baldoni et al., 2010).

IODA (Kubera et al., 2008) provides a conceptual framework for designing a MAS by focusing on the interactions. In IODA, all the entities –even the environment– are considered as agents. An interaction between two agents is triggered by a precondition in order to achieve a post-condition.

The MERCURIO (Baldoni et al., 2010) framework allows to explicitly represent not only the agents, but also the rules, the conventions, the resources, the tools, and the services. Hence, it provides a formal model of interactions between the different dimensions of the MAS.

3.1.4 The Organization Dimension

The Organization dimension deals with the collective behavior of the system. Thus, this dimension considers the cooperation, coordination and regulation mechanisms. In fact, such a collective behavior results from the interactions between the different entities of the MAS and then it can be different to the sum of each individuals' behavior. It can lead to the emergence of unforeseen effects, either positives or negatives.

The way the roles, the relationships, and the authority structures of the MAS are defined by the organization can be various in nature (Horling and Lesser, 2004): flat organizations, hierarchies, holarchies, teams, coalitions, open societies...Further more, the organization can be the result of a bottom-up process (Di Marzo Serugendo et al., 2006; Glaser and Morignot, 1997) –i.e. resulting from the agent's

[1]FIPA website: `http://www.fipa.org/`

[2]KSE website: `http://www-ksl.stanford.edu/knowledge-sharing/`

[3]`http://jade.tilab.com/`

interactions– or a top-down design (Horling and Lesser, 2004; Jennings, 2001) –i.e. defined at the conception phase.

Thus, it is possible to categorize organizations along two dimensions (Picard et al., 2009): (i) the organization (un)awareness of the agents and (ii) the implicit/explicit definition of the organization. Therefore, four categories of MAS can be distinguished from the organization point of view (see Table 3.1).

In emergent MAS and in AOSE, agent have no representation of the global system's objectives. Nevertheless, in AOSE, agents are designed in a top-down manner, after having described the collective behavior (Bresciani et al., 2004; Kubera et al., 2008).

On the contrary, an emergent MAS consists of designing and parametrize the agents' behavior in order to lead to a collective behavior, generally too complex to specify (Capera et al., 2003; Anthony and Mccann, 2007; Di Marzo Serugendo et al., 2006).

In coalition oriented MAS and in organization oriented MAS, agents conscientiously collaborate with the other agents in order to achieve collective goals. On the one hand, in coalition oriented MAS, the agents regulate themselves and, possibly, choose a leader among themselves (Carr et al., 2009; Overeinder et al., 2002).

On the other hand, in organization oriented MAS, an explicit organizational entity controls and regulates the agents' behaviors. Several organizational frameworks have proposed, including STEAM (Tambe, 1997), ISLANDER (Esteva et al., 2001), $\mathcal{M}$OISE (Hübner and Boissier, 2002), OperA (Dignum, 2004), AGR (Ferber et al., 2004), or TÆMS (Lesser et al., 2004). These frameworks enable to explicitly define the organizational specifications along several concepts, that we can classify along three dimensions:

Social goals express the global objectives and sub-objectives of the MAS. It usually consists of an AND-OR tree decomposition of a main goal into sub-goals that should be satisfied by the whole MAS (Tambe, 1997; Hübner and Boissier, 2002; Lesser et al., 2004). At the organizational level, these goals and plans indicate the MAS what to do rather than how to do.

Roles are an abstract representation of the position of the agents in the organization (Hübner and Boissier, 2002; Ferber et al., 2004; Dignum, 2004). Typically, a role represents a functionality, or a set of

	Organization Unawareness	Organization Awareness
Implicit Organization Definition	Emergent Organization MAS	Coalition Oriented MAS
Explicit Organization Definition	Agent Oriented Software Engineering	Organization Oriented MAS

Table 3.1: Classification of Multi-Agent Systems with respect to the organization definition and awareness. (Picard et al., 2009)

capabilities, of the system the agents will be responsible for. Such a role defines the relationships between the agents committed to this role and the others: acquaintance, communication, groups and hierarchy between the roles.

Norms define a deontic right –i.e. permission, obligation, interdiction– regulating the actions of the agents in the MAS (Esteva et al., 2001; Dignum, 2004; Hübner and Boissier, 2002).

Further in this chapter, a more detailed analysis of the benefits and drawbacks of the different approaches is done with respect to the governance perspective: the impact of the organization on the expressiveness (Section 3.2) and on the adaptation mechanisms (Section 3.3) of the MAS.

3.1.5 Relevance of the Multi-Agent Oriented Programming approach for Governance

These four dimensions of Multi-Agent Systems are of great interests for a governance system. In fact, they enable the separation of different concerns which will enable the designer to focus on the different aspects of the governance: expressing the governance strategy and objectives, monitoring and controlling a system, reasonning on the governed system's behavior and on the governance's performance itself.

With these objectives in mind, we focus our litterature review following the two following questions: "how to define and implement the control of a legacy system ?" (Section 3.2) and "how to adapt this control ?" (Section 3.3).

3.2 Multi-Agent System from the Control perspective

A governance system requires to be able to control the system with respect to the governance strategy and objectives. This control includes the following steps: (i) define the global objectives of the system, (ii) set-up/configure the system following the objectives, (iii) monitor the system and check its compliance with the defined objectives, (iv) decide if adaptation of the legacy system is needed ("what?" and "how?"), (v) apply the decision and back to step (iii) . Therefore, this section studies how such a strategy and such global objectives can be defined in a MAS. Further more, this section also addresses the problem of applying this strategy to a legacy system, i.e. an external system encapsulating the business logic.

As stated in Section 1.1.1, two kinds of requirements are used to define the objectives of the system: *functional requirements* and *non-functional requirements*. While the former describes the tasks and functions required to make the system realize its purpose, the latter refers to properties the system should hold when performing its tasks (Service Level Agreements (SLAs)).

3.2.1 Goal Oriented Requirement Engineering

A key concept in MAS is the concept of *goal*. The Goal Oriented Requirement Engineering (GORE) (Yu, 1997) provides a methodology to design system requirements based on this concept. In fact, using goals as a central element of the system allows the designer to not only to define *"what are the requirements"* but also *"the reason why they are required"*.

Hence, the ***i**** framework (Yu, 1997) is based on a *means-end* analysis. This framework is composed on three types of components: (i) *actors* are active entities capable of acting processes to achieve the system's goals, (ii) *intentional elements* representing entities of the system to be manipulated by the actors and (iii) *dependencies* describing why the actors need to manipulate the intentional elements. This intentional elements, in turn, can be of different natures: (i) *resources* to be furnished, (ii) *tasks* to be performed, (iii) *goals* to be achieved and (iv) *soft-goals* to be statisficed –i.e. goals which achievement cannot be formally described but giving a direction to its satisfaction.

The system requirements are defined within a two-step process, in which the intentional concepts are exploited:

1. The *Strategic Dependency* of the system consists of the identification of the actors of the system and the definition of the intentional relationships between them. For example, Figure 3.3 shows that the actor `Meeting Initiator` depends on the actor `Meeting Participant` to achieve the goal `Attends Meeting` and also to get the resources `Exclusion Dates`, `Preferred Dates` and `Agreements`.

2. The *Strategic Rationale* is the definition of each actor's internal intentional elements and dependencies. For example, in Figure 3.3, the process `Schedule Meeting` contributes to the achievement of the goal `Meeting Be Scheduled` and reduces the satisfaction of the soft-goals `Quick` and `Low Effort`. The process is decomposed into three intentional elements: the processes `Obtain Available Dates` and `Obtain Agreements` (which depend on external resources) and the achievement of the goal `Find Suitable Slot`.

Such a methodology focuses on the definition of early requirements, that is, a high level definition of the system. This definition helps the designers to build a system (e.g. a MAS) following these requirements. Nevertheless, it does not specify how the requirements should be specified and implemented in the development process. Further more, it raises the problem of the propagation of new or modified requirements to the agents.

3.2.2 Agent-based Control

Bresciani et al. (2004) propose a methodology to design a MAS based on ***i****. It propagates the ***i**** concepts through the whole design of the MAS –i.e. until the detailed design of each agent– in order to ensure that the agents design follows the requirements. Such a process can be divided into three phases: (i) the requirement analysis, (ii) the architectural design, (iii) the agent design, and (iv) the implementation.

1. The *requirement analysis* is decoupled into the early and the late phases. First, the early requirements phase consists of modeling the actors, goals and capabilities of the MAS. Then, the late requirements phase describes the system-to-be: (i) the dependency between an actor and the others corresponding to the Functional and Non Functional Requirements (NFRs) of the system, and (ii) their decomposition into goals, sub-goals and plans.

2. The *architectural design* details the previous system definition. First, *new actors* are defined corresponding to the analysis of different level of abstraction: (i) sub-actors for the delegation of the sub-goals, (ii) new actors depending on the architectural choices (e.g. brokers, coordinators...), and (iii) actors to improve the satisfaction to the NFRs. Then, the *capabilities* needed by the actors to fulfill their goals and plans are identified based on (i) the dependencies between a role and a goal

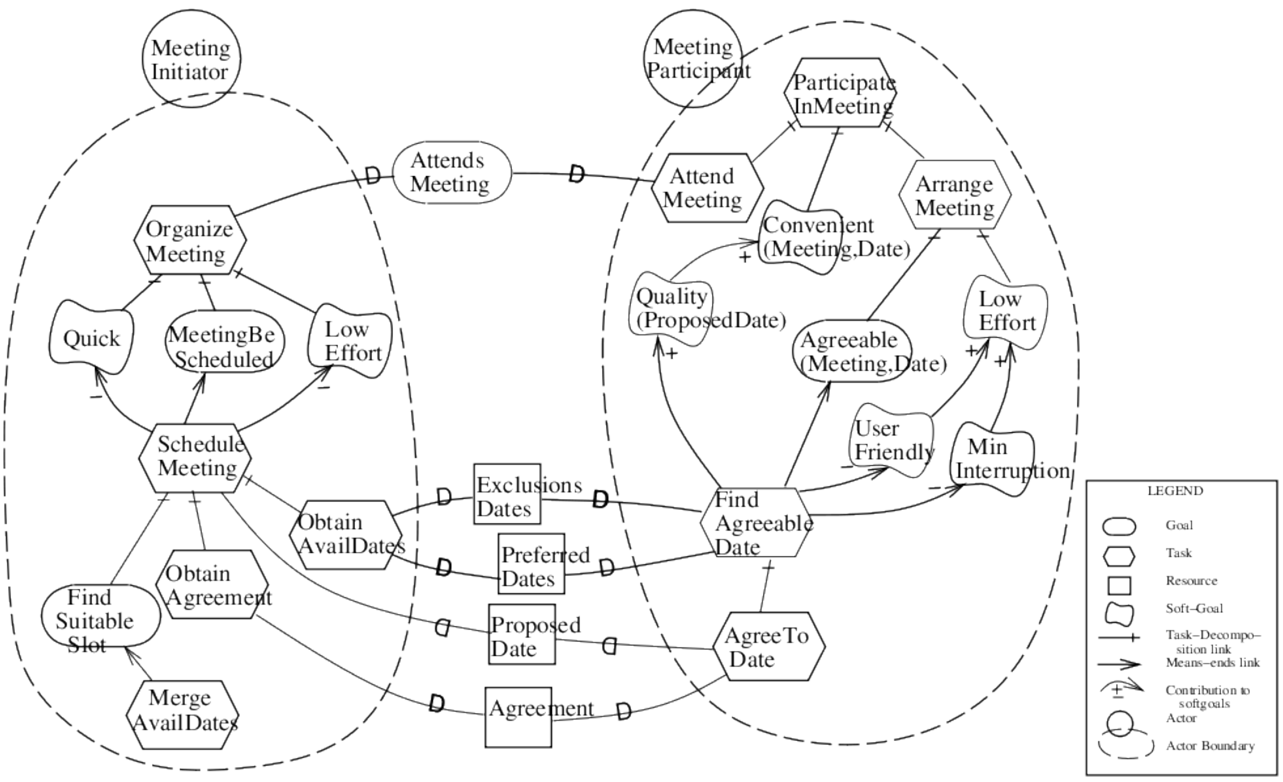


Figure 3.3: Early Requirement definition using the ***i**** framework

(i.e. get- and provide- capabilities), and (ii) the actions performed by the actor itself (e.g. aggregate). Finally, the set of *agent types* are defined and are assigned to one or more capabilities, each capability can be assigned to several agent types.

3. The *design of the agents* –capabilities, plans and interactions– is detailed using AUML (Bauer et al., 2001). Each *capability* is represented as an activity diagram. Then, each *plan* node of the capability's activity diagram is, in turn, detailed in an activity diagram. Finally, *interactions* between the agents –with other agents or resources– are modeled by AUML sequence diagrams.

4. The *implementation* is done following the detailed design using a BDI agent platform –JACK (Busetta et al., 2000). It provides a direct mapping with the capabilities and plans of the previous step.

Following this process ensures the agents will follow the requirements designed based on the governance objectives. Nevertheless, the requirements defined during the design phase are not yet available during run-time. Thus the agents can only follow them, but not reason about them.

The ADELFE methodology (Picard and Gleizes, 2004) helps the designer to develop MAS from the requirements to the deployment to identify the adaptive parts of the system, i.e. the agents.

This identification is based on the AMAS theory (Capera et al., 2003): agents are cooperative ones –i.e. they try to solve a global problem cooperatively. These agents ignore the global objectives, but focus on individual purposes. That is, the global function is emerging from the agents' interactions.

To do so, agents detect so called Non-Cooperative Situations which should be avoided, from each agent's point of view. A Non-Cooperative Situation expresses a constraint on the system. Hence, this approach enables to build an adaptive and reactive governance for complex systems by defining the requirements at the agent's level.

The main advantage of the ADELFE approach is it keeps the search-space wide by expressing the parts to avoid only. However, its main drawback from a governance perspective is that the global function is not explicit at run-time but can only be observed as an emergent function. This leads to less predictibility of the governance.

3.2.3 Interaction-based Control

As the interaction dimension is transverse to the other dimensions of the MAS, it is a key part to observe and control the whole system. With the development of the Service Oriented Architecture (SOA) and e-commerce applications, controlling the interactions between open, decentralized and dynamics systems has become a crucial part.

Hewitt (1977) introduced the concept of controlling decentralized systems through the communication mechanisms between the different actors –e.g. agents[4]– of the system. Hewitt demonstrate that message passing could be used to analyze the control structure patterns in such a decentralized system.

Carron et al. (1999) propose TACL –an Agent Communication Language (ACL) enriched with temporal data. This allow to define an time limit constraint to perform the act. Hence, it is possible to control the agent's interactions according to some requirements explicitly.

Such a language enables to express constraints within the dialog between the agents. Nevertheless, it does not allow to constraint the interactions themselves.

Interaction protocols have been proposed to tackle this issue. The FIPA (FIPA, 2000) provides a set of protocols as a referential for common MAS dialogs. For example, the Contract Net Protocol (CNP) is an interaction template for two agents –the *initiator* and the *participant*– to request an action and negotiate the conditions of the agreement.

The AUML (Bauer et al., 2001) provides an interaction description language. It extends UML sequence diagrams with to define *protocol diagrams*, that is, the succession of interactions between the agents to occur during a process (see Figure 3.4).

These two approach allow to define a protocol of interactions at the design level. They enable to design various and complex protocols. But they do not permit to control the interactions at run-time.

Populaire et al. (1993) use a transition diagram to specify and control the interaction protocol at run-time. Each transition specify constrains on the interaction between two state, such as the type of interaction, or beliefs of the agent before it can initiate the interaction.

[4]The concept of *agent* was introduced latter, in the 80s and 90s, merging works from several scientific communities.

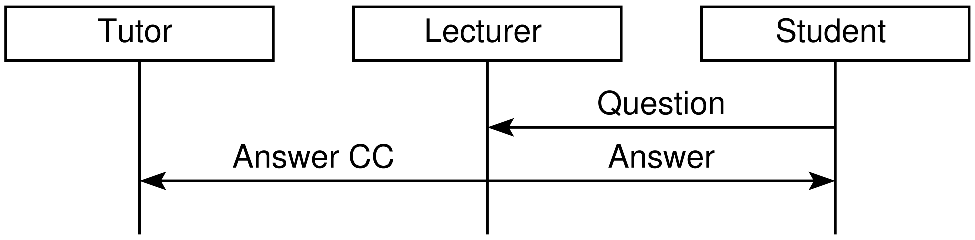


Figure 3.4: Graphical representation of a simple *AUML Protocol Diagram* representing between three agents (Tutor, Lecturer and Student).

This allow to define simple (with few states), yet generic, interaction protocols. Nonetheless, big transition diagrams induce more complexity. Though, the agents can compose several interaction protocols to implement more complex protocols.

As described in Section 3.1.3, the IODA framework (Kubera et al., 2008) allows the system designer to define the pre- and post-conditions for each interaction. It is then, possible to specify an interaction protocol, by using the achievement of an interaction as a trigger condition for the next interaction.

Another approach is to define *conversation policies* (Greaves et al., 2000). A conversation policy is declarative specification for governing interactions between the agents. It provides proper semantic to define the interaction decision processes of the agents to interact, taking account of their intentions, the context defined by the previous messages and the context defined the previous interactions. This simplifies the agent decision process and offers better guarantees for a common collective behavior.

Controlling interaction allows to specify requirements about the overall system. But, the bigger and the more open, dynamics and decentralized the MAS is, the more complex and difficult it is to specify and control all the interactions.

Further more, to our knowledge, the litterature in interaction-based control focus more on the system's functioning rather than the system's requirements. Though, some of them provide a mean for supporting the requirement control within the MAS. Thus, interaction appears to be a part of the control process.

In the next section, we focus multi-agent organizations that provide another mean to specify rules governing the MAS.

3.2.4 Organization-based Control

Brazier et al. (2010) uses a MAS in order to deal with the dynamics of Service Oriented Architecture (SOA). In particular, the proposed system uses the software agents capabilities for the run-time establishment of Service Level Agreements (SLAs) and the control of the Quality of Service (QoS).

In this model, *organizations* are used to define the means to achieve the global goals and facilitate the coordination and the flow of information. The business model is represented by normative relationships understandable by the agents. Hence, the organization correspond to the definition of the functional requirements of the system.

Then, agents achieve run-time coordination between the services, following the organizational definition. They invoke the services they encapsulate to achieve their goals. Agents also have an internal representation of the SLA corresponding the encapsulated services. When a service fails to comply with the SLA, the agent tries to find alternative services. Agents are responsible for managing the non-functional requirements, but still have to deal with the legacy process.

Norm-based Control. In a similar approach, Pitt et al. (2006) uses normative agents to coordinate services in a Multimedia Ad-Hoc Network (MANET). Each agent manages a resource of the network. But here the norms, based on Event Calculus (Miller and Shanahan, 1999), define protocol for agents to accept/deny the sharing of the resource (*e.g. start a vote, grant an agent the veto power...*). This decision is made collectively by the agents who are involved in applications using the requested resource –i.e. the resource is used in the flowgraph of the application.

Information and access about the network is given at a lower level layer of the governance system: a central node (or a Cluster Head), called Multimedia Newtork Support Platform. It hosts several control services such as *Synchronization*, *QoS*, *Call Admission Control)...*The individual parts' QoS is not computed at the agent level but at the environment level. Based on this information, the agents implement the QoS Distribution Monitoring algorithm: the global QoS is decomposed for each part of the flowgraph, each part of the decomposition is monitor by an agent. The service platform also computes the shared flowgraph and transmits the proposition to the MAS for deliberation.

Thus, the agents can focus on the governance decisions. However, this model gives no formal definition of the SLA and offers a poor separation of the functional and non-functional requirements.

Herssens et al. (2010) show that norms are relevant for the SLA definition. It propose to describe each SLA as a set of norms. In this case, requirements define with the WSLA language are translated to norms in the NoA language.

Stakeholders are represented by normative agents that comply to the norms. This compliance can be checked using two principles: (i) mutual obligations and (ii) supervised interactions . The mutual obligations principle is similar to a commercial exchange: an agent will do its best to comply to the contract because its effort will be compensated by the other agent's service (or payment).

But as the transaction is diachronic, this mechanism doesn't prevent from opportunistic behavior from the last agent, even though the first has accomplished its mission well. Therefore, a third-party controller can be involved in the transaction. It will monitor the services provided, compute the QoS values, check their compliance to the SLA and eventually sanction violations of the SLA.

Although it does not provides a model for the functional requirement, such an approach gives a clear representation of the non-functional requirements for both the designer and the agents.

Similarly, Sycara et al. (2009) use normative policies to define the functional requirements for controlling collaboration between human groups in constrained environment (e.g. humanitarian aid).

Collaborative interventions are constrained using deontic relations (norms). Thanks to these norms, the agents can detect the violation of policies. This policy reasoning is performed through a set of rules such as an expert system. In the given use case, this detection is used to interfere with communication to censor the violating parts of the messages.

Role-based Control. The concept of role is widely used in Multi-Agent Organizations (Demazeau, 1995; Ferber et al., 2004; Horling and Lesser, 2004; Jennings, 2001). At a similar level of abstraction to the UML Use Case, the role is used to define the responsibilities, functions and interactions with the other roles of a program in an organization.

That is, the role is similar to an interface (in Object Oriented Design) for the agents: agents comply to their roles but the functions can be done/implemented in different ways. Further more, the agent can decide to/not to fulfill its role's functionalities. For example, it can refuse to respond to a request in order to achieve a more important task or to save resources.

Chan and Sterling (2003) propose to use the concept of *role* in order to define the *functional requirements*, even for non agent-based systems. The roles define the functionalities derived from the requirements in terms of *Responsibilities*, *Initiatives* and *Facilities*.

A responsibility is a function the agent should ensure regarding environmental changes. An initiative is a function the agent should ensure proactively. And a facility is a function the agent should ensure when requested. These functionalities are parametrized with QoS value, in particular to define the deadline to achieve the functionality (i.e. maximal latency).

This approach provides a clear separation of the functional and non-functional requirements. Nevertheless, this definition of roles lacks of flexibility. In fact, the functionalities are attached to the role which makes the role difficult to evolve and the functionalities poorly reusable.

The $\mathcal{M}$OISE approach. To tackle the problem mentioned previously, the $\mathcal{M}$OISE framework (Hübner and Boissier, 2002) provides a modular approach to define an organization along three dimensions: (i) the structural dimension, (ii) the functional dimension and (iii) the normative dimension .

The structure describes the organization as groups composed of roles involved in the MAS. For each role, a cardinality of agent committed to the role is specified. The structural dimension also defines the acquaintance, communication, compatibility and hierarchical relationships between the roles.

Then, the functional dimension define the objectives of the MAS as a set of social schemes. A scheme is a tree decomposition of a social goal into sub-goals which a group will be responsible to satisfied. Consistent set of goals are grouped into missions. Both goals and missions can be qualified with the cardinality of agent and a time to fulfill them.

Finally, norms are used to assign missions to the roles. A norm is composed of (i) a context condition that triggers the norms, (ii) a deontic relation (i.e. permission or obligation) committing (iii) a mission to (iv) a role. It can also be specified a time for the agents to take in charge the mission, before the norm is considered to be violated.

The three dimensions of the $\mathcal{M}$OISE framework provide more flexibility to the organization definition. In fact, the same mission can be assigned to different roles under different conditions. Further more, it is possible to redefine the structure in order to enforce/relax social constraints. In the same way, functional schemes can be extended/reduced to change the agents' degree of liberty. In fact, the agent can reason on the organizational constraint and decide whether to comply to or deny them.

3.2.5 Environment-based control

In most of theses approaches, agents directly encapsulate the legacy process. This means they have to consider in the mean time both the applicative design and the non-functional requirements. Yet, applicative processes and events can be represented in the environment (Demazeau, 1995; Jennings, 2001). Even though it is the case in Sycara et al. (2009) and in Pitt et al. (2006), both of these approaches do not provide a formal abstraction of these processes.

Hybrid Agent/Component Architecture. An hybrid agent-based and component-based software engineering approach integration is proposed in (Dragone et al., 2011). The SoSAA framework separates the application and infrastructure concerns using both the Agent- and Component-based approaches.

Applicative functionalities are implemented by components-based modules. The agents are used to control these components through an adapter. The adapter enables to (un)load, (de)activate, (un)wire and configure the components. It also provides information to the agents about the available components and their interfaces, their performance and events occurring.

Thanks to these information, coordination and BDI reasoning on the legacy system is done at an upper level by the agents.

Agents & Artifacts Meta-Model. An other approach is to use the Agents & Artifacts (A&A) meta-model to provide an abstraction of the business processes (Omicini et al., 2008) and implemented by the CArtAgO framework.

Artifacts can be shared and observed by the agents within a distributed environment. Figure 3.5 shows a representation of an artifact in CArtAgO. It is composed of a description of (i) *signals* the artifact can send to the agents, (ii) *properties* observable by the agents, (iii) *operations* the agents can invoke, and (iv) *linkable operations* which allow to the agents connect several artifact together . Agents can create, duplicate, activate the artifacts and call operation provided by the artifact to enact the required process.

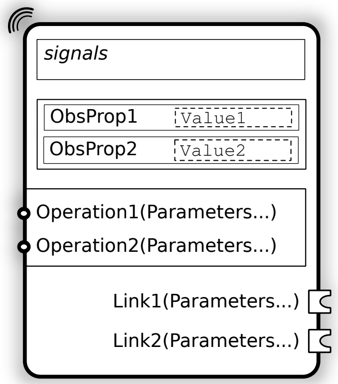


Figure 3.5: Representation of an Artifact following the Agents & Artifacts meta-model. (Omicini et al., 2008)

This approach has been validated by several works: in Minotti et al. (2011) for integrating MAS in Web 2.0 applications, in Santi et al. (2011) to encapsulate Android components as artifacts and in Piunti et al. (2009) to interface the MAS with SOAP-based Web Service. Using the approach given by Piunti et al. (2009), the artifacts generate SOAP interfaces which can be used to wrap existing Web Services, giving the control to the MAS.

Thus, the artifacts represent an interface with exogenous systems. Therefore agents can concentrate on their tasks without any details on the legacy processes and protocols. Further more, Hübner et al. (2010) provide a CArtAgO support for $\mathcal{M}$OISE organizational artifacts. The A&A meta-model brings good separation of concerns in the MAS by giving an abstraction of the system of the agents.

3.2.6 Synthesis on MAS-based Control

In the previous sections, we have studied various means to define and specify the governance objectives and control their satisfaction. We can now synthesize them in order to keep the main principles for our governance model.

Table 3.2 reports these approaches and compares the way the functional and non-functional requirements are specified and ensured. We can separate two approaches.

On the one hand, the AOSE approaches focus on the agent's design from the requirement definition (Bresciani et al., 2004; Dragone et al., 2011; Picard and Gleizes, 2004; Yu, 1997).

On the other hand, organization oriented MAS use a declarative approach for the requirement definition, which is enabled by normative agents (Brazier et al., 2010; Chan and Sterling, 2003; Herssens et al., 2010; Hübner and Boissier, 2002; Pitt et al., 2006; Sycara et al., 2009). Such a declarative approach benefits from two main advantages: (i) the requirement expression is easier to translate from the human language, and (ii) the separation between the "what" (organization) and the "how" (agents) concerns is clearer.

A second aspect of the MAS governance is how the business process is controlled. Several work embed the business process directly into the agents themselves (Brazier et al., 2010; Bresciani et al., 2004; Chan and Sterling, 2003; Herssens et al., 2010; Pitt et al., 2006).

For others, like Sycara et al. (2009), the business process is supported by an external (from the MAS) system. Picard and Gleizes (2004) distinguish the active entities, embedded into the agents, and the passive entities which are located in the environment.

For Dragone et al. (2011), the business processes –legacy components– is clearly separated from the reasoning entities –agents. According to Omicini et al. (2008) the environment is privileged abstraction for the "non-agents" entities. That is, the environment should not only embed the business processes but also abstract and interface them, so that the agents can monitor and manipulate them more easily.

To summarize, we can differentiate three level of governance control in a MAS, to which we can add the interactions that enable the control process:

1. the requirement definition and specification using the *organization*,
2. the requirements' satisfaction analysis and reasoning concerns by the *agents*, and
3. the business processes' abstraction through *artifacts* providing monitoring and controlling functions.

As explained in Chapter 1, the governance process should include a refinement phase. Such a refinement consists of enforcing, relaxing, adding or removing some of the system's requirements. This kind of process is supported in MAS by adaptation mechanisms. Therefore, we will study these mechanisms in the next section.

Further more, we have identified, in this section, that the organization was the most suited dimension of a MAS to declare and carry the system requirements. So, our literature review will focus especially on the adaptation of the MAS organization.

	Functional Requirements	Non-Functional Requirements
Brazier et al. (2010)	Organizational choreography	Agent-governed coordination
Bresciani et al. (2004)	Base Capabilities	Additional Capabilities
Chan and Sterling (2003)	Role's Responsibilities, Initiatives, Facilities	Constraint parameters
Carron et al. (1999)	–	Enriched ACL with Temporal Data
Dragone et al. (2011)	Application Agents	Infrastructures Agents
FIPA (2000)	AUML Protocol Templates	–
Greaves et al. (2000)	Declarative Conversation Policies	Declarative Conversation Policies
Herssens et al. (2010)	–	Norm-defined SLA
Hewitt (1977)	Message Passing Patterns	–
Hübner and Boissier (2002)	MAS Structure ; Social Schemes	Cardinality, Time To Fulfill
Kubera et al. (2008)	Interaction pre-/post-condition	–
Picard and Gleizes (2004)	Emerge from Agents' Cooperation	Non Cooperative Situations Resolution
Populaire et al. (1993)	Constrained Transition Diagrams	Constraint parameters
Pitt et al. (2006)	Agent belief ; Shared flowgraph by lower layer	Norm-governed agents implement QoS distribution monitoring
Sycara et al. (2009)	Normative Policies ; Censor Agents	–
Yu (1997)	Goals	Soft-Goals

Table 3.2: Requirements in Multi-Agent Systems

3.3 Multi-Agent System from the Adaptation perspective

The previous section studied how the governance can be specified and implemented by a MAS. We now focus on the process that enables the governance adaptation, such as the CobiT and SOA governance cycle. Such cycling process constantly (i) monitors the system's performances to (ii) analyze the governance's efficiency and (iii) refine it to increase the benefits to, finally, (iv) reconfigure the system accordingly (see Section 1.1, p. 10). In MAS, such an adaptation cycle is performed through the so-called *reorganization* process.

In fact, it is not sufficient to be able to define and implement the governance strategies and policies. As the governed system evolves, the governance should evolve too. Therefore, the governance management system should be able to reorganize and adapt itself.

In Section 3.2, we have shown that an explicit organization design fitted better to the definition of governance objectives. Nevertheless, it should be possible to make these objectives evolve while the governance strategy evolve.

We first highlight the motivations which conducted researchers to work on MAS reorganization in Section 3.3.1. Then, we study how it is possible to reason about the MAS in order to reorganize it (Section 3.3.2) and how such process can be supported in MAS (Section 3.3.3). Finally, 3.3.4 draws some known issues with respect to the reorganization.

3.3.1 Motivations for MAS Reorganization

In Glaser and Morignot (1997), the organization changes due to agents migration –e.g. integration/departure of an agent in/from the organization. On the one hand, the new agent is accepted only if it increases the utility of the whole MAS. On the other hand, an agent joins the society in order to increase its own utility. The authors consider the commitment to a role as an agreement of a new convention between the agent and the society.

They distinguish three different variation of the concept of role. First, the *Desired Role* is the role the Agent wants to play. Then, the *Expected Role* is the role the society expect the agent to play. Finally, the *Committed Role* is the role an agent have committed to play within a society.

The set of committed roles constitute a so-called *convention*, i.e. an implicit agreement on a distribution of roles. Such a convention may emerge from the agents' activity providing that they have a background knowledge about possible structures of the society.

Such a concept of reorganization requires the following elements to be defined. First, the motivations of a society to accept/refuse an agent must be expressed. It is performed through two concurrent processes: (i) the members of the society punish/award the candidate agents until they have adapted to the convention, and (ii) the society is adapted to accept the candidate, and the latter rewards the members. Then, it is necessary to express the motivations of an agent to accept/refuse a role. And, finally, both the agent and the society should be given a utility function pertaining the benefits of accepting/rejecting a commitment.

Following this view, the key for reorganization lies in the contradiction between the individual objectives and the global objectives: i.e. while the organization defines the macro-level behavior of the MAS, autonomous agents at the micro-level, can decide whether to follow the rules or overpass them. Thus, the actual running MAS results from a micro-macro loop of compromises for which both levels benefit.

Oren et al. (2007) also support this point of view, considering that the agents' loss might serve the MAS welfare. Further more, the authors reveal another contradiction residing in the collaboration and competition relationships between the agents, such as human relationships. In fact, an autonomous agent in a MAS (i) have its own goals to fulfill, but (ii) must interact with other agents to fulfill them. So, the agent must balance its own welfare and the cost of collaboration.

3.3.2 Agents' reasonning for Reorganization

In the previous section, we have shown the difference between the agent point of view and the organization point of view. We, now, focus on the expression of the agents' interests and their ability to adapt and reorganize the MAS.

In order to adapt the organization of the MAS, an agent must be able reason about it. In fact, not all the agents are able to reorganize. Glaser and Morignot (1997) identify four levels in the agent's reasoning. First, the *reactive level* consists of basic actions based on the agent's perception. Then, the *cognitive level* contains a set of individual action plans for achieving non-trivial goals, but cannot handle conflicts which may arise with other agents. This requires protocols to cooperate with other agents, to build joint plans and to solve conflicts defined at the *cooperative level* to fulfill a static role in the organization. Finally, the *social level* specify the agent's behavior to agree with other agents about the role assignments and the competences to reorganize.

First, we describe how an organization can be modified (Section 3.3.2.1), and then we study how the agents of a MAS collectively decide to reorganize (Section 3.3.2.2) or not (Section 3.3.2.3).

3.3.2.1 What to Reorganize

According to Hübner, J. F. and Sichman, J. S. and Boissier, O. (2004), a reorganization can be considered at two levels: (i) the *Organizational Specification (OS)*, or (ii) the *Organizational Entity (OE).*

1. The OS level consist of a description of the structure, the functionalities and the norms of a MAS. Hence, a reorganization can consist of the redefinition of one or several of these dimensions. For example, in Glaser and Morignot (1997), the agents change the organization's role to meet or to get closer to their expected role, and Bora and Dikenelli (2009) propose to duplicate the roles, or sub-roles, corresponding to critical parts of the system. Hübner, J. F. and Sichman, J. S. and Boissier, O. (2004) use several expert agents which propose a single kind of change –a new structure of roles (e.g. roles' cardinality) or new plans (e.g. goal sequences)– that are selected using a Q-Learning algorithm.

2. The OE level is an instance of the OS. That is, a configuration of the agents that respects or violate the OS constraints but is not explicitly specified. In fact, the agents are free to decide how they fulfill their roles and missions and whether to fulfill the OS or violate it (Hübner and Boissier, 2002). For example, Kota et al. (2012) propose a reorganization mechanism in which the agents evaluate their

load to delegate tasks to other agents (and to accept the tasks) to fulfill the collective plan. In Oren et al. (2007), the agents decide if the punishment policies should be applied or not or negotiated. As a matter of fact, Morandini et al. (2009) identify three types OE adaptation: (i) the *tuning* consist for an agent to change its own behavior, (ii) the *reorganization*, when an agent changes a partnership (i.e. interactions with other agents), and (iii) the *evolution* of the organization, by which new agents are created or deleted.

To synthesize, we can observe that these two levels of reorganization address different kind of problems. Thus, we can prioritize the type of reorganization using the following hierarchy: (i) an OE level reorganization addresses a punctual need for reorganization, where the adaptation is less formal, which brings flexibility to the OS, while (ii) an OS level reorganization requires a more important reflexion as it will impact the formal specification of the system in the long term.

3.3.2.2 Reasons to Reorganize

The reorganization usually is the consequence of an inappropriate organization. But, such a case can be expressed in different manners.

Hübner, J. F. and Sichman, J. S. and Boissier, O. (2004) identifies three types of adaptation processes, ordered by the degree of the agents' liberty: (i) the *predefined adaptation*, i.e. the reorganizations are planned in advance whatever happens to the system, (ii) the *controlled adaptation*, i.e. the reorganization processes are specified according to some conditions and is controlled, in a top-down manner, by the MAS or an external entity, and (iii) the *emergent adaptation*, i.e. the agents trigger the reorganizations themselves, in a bottom-up manner, without any control. The more liberty the agents have, the less control the designer have on the result of the reorganization (Picard et al., 2009).

OS level. The agents can control themselves the OS level reorganization. In Glaser and Morignot (1997) the (OS level) reorganization is provoked by an agent which expect to commit to a role that does not fit the organizational specification. The organization, in turn, accept or refuse such a change according to the collective utility it brings.

In Bora and Dikenelli (2009) the (OS level) reorganization is performed by the agents to ensure the fault tolerance of a critical role. An agent consider its role as critical if (i) it cannot access to all the required resources, or (ii) there is a conflict between different roles the agent enacts.

OE level. Addressing the OE level reorganization, agents are directly concerned for the decision to apply or violate the OS.

In Oren et al. (2007), the reorganization consists of the relaxation of social norms. When a contract agreement is not properly fulfilled, the wronged agent can claim compensation from the other through an enforcement mechanism. Nevertheless, claiming compensation also have a cost: e.g. resources to prove the wronged, worsening of future collaborations. . . That is why before starting a contract enforcement, the authors suggest that the agent should evaluate possible gain and loss. Same applies to the accused agent before contesting the enforcement.

In the AMAS theory (Capera et al. (2003), see Section 2.1.4, p. 46) the reorganization is triggered by environmental causes. An agent tries to adapt the organization when it locally detects a so-called Non-Cooperative-Situation (NCS). Such an NCS can be either (i) a signal perceived from environment which is not understood or ambiguous, (ii) a perceived information that does not produce any activity of the agent, or (iii) conclusions which not useful to the other agents.

Such a bottom-up mechanism is used in Morandini et al. (2009) to enable the agent to reorganize in a top-down MAS. The authors propose an extension of the TROPOS framework with the ADELFE methodology (both are presented in Section 3.2.2, p. 68). The former allows to design the nominal behavior of the agents inferred by the global objectives' decomposition, while the latter is used to design the exceptional behavior. When failures in the plans are detected (NCS), the agents react to trigger a reorganization.

We can observe that the agents, or the controller entity, need to analyze the situation in order to trigger a reorganization. This situation may vary depending on the goals of the MAS.

So, the designer has to identify the conditions for which the organization might not fit to the situation and program the agents according to these conditions. He also has to provide tools so the agents can effectively and efficiently be aware of the reorganization conditions and impacts. We will discuss this point in Section 3.3.3.

Nevertheless, too much reorganization may result in an unstable system: if the objectives or the structure of the system always change, it will be inefficient. Inertia in the organization is thus necessary in order to ensure the MAS to realize its tasks.

3.3.2.3 Reasons not to Reorganize

In this section, we focus on the means to stabilize the organization. That is, how the agent reason in order not to reorganize all the time.

Reorganization impact. First, the agents have to take into account the impact of the reorganization. In fact, adapting the organization not always result in better performance.

In Kota et al. (2012), the agents evaluate the performance as a profit function. That is, the performance of the organization is the difference between the sum of the rewards –i.e. the fulfilled mission– and the computational cost –i.e. use of resources and network. In this case, such a performance is calculated locally by each agent, based on their partial knowledge.

But Alberola et al. (2012) use a `Reorganization Facilitator` service to which the possible adaptions are submitted. It evaluates the costs and benefits of adaptation processes, depending on the moment of the execution, and triggers it or not.

Reorganization Cost. In fact, it is also necessary to evaluate the cost of the reorganization itself.

For Alberola et al. (2012), such a cost is evaluated based on the organization transitions. In particular, it uses a use multi-transition deliberation mechanism that calculates transitions in different dimensions (roles, services, relationships, agent population) to other organizations with high expected utility based on the cost for transition to these organizations. Then, it to decide which transitions to perform and in which order. In this case, all the possible transitions are evaluated centrally.

Another approach is to distribute the decision over the agents (Kota et al., 2012). Before performing the reorganization, the agents evaluates the collective utility of the

Reasoning Cost. As such a computation also have cost, the agents do not evaluate all the possible changes with all their peers. So, the authors propose a meta-reasoning for reorganization: the agents only evaluate k relationships chosen randomly as candidates for reorganization. This k value may vary depending on the agent's load: (i) it will consider at least one candidate ($k = k_{min} = 1$), if it is not overloaded, (ii) it will evaluate as much candidates as it can ($k = free\_capacity$), if not, (iii) it will force to evaluate some candidates, depending on the previous iteration ($k = success\_ratio$).

This way, it ensures the agents will reorganize when the organization is inefficient, even if they are overloaded, but it moderates the reorganization reasoning cost, in the mean time.

3.3.3 Support for MAS Reorganization

In the previous section, we have studied the needs and causes for the organization to be adapted or not, and how the agents could reason about it.

We now study how the MAS can support such a reorganization as a micro-macro loop. In fact, this is not a trivial problem. Once the MAS is running, the agents should be able to detect the modifications of the organization.

That is, the new specification or the new entity should be communicated to the concerned agents. In turn, this agents should adapt their behavior according to the new organization.

In this section, we focus on the tools and methodologies that contribute to facilitating the reorganization. We study three level of reorganization support: 1. the organizational support, 2. the agent-level support, and 3. the environment support.

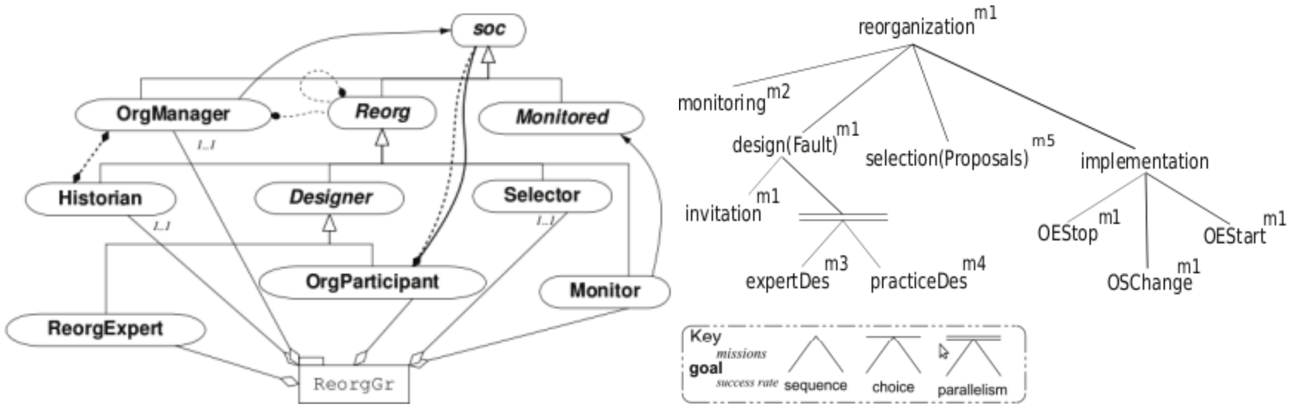
We do not address the interaction support for reorganization as it is transverse to the other dimension. In fact, reorganization is the result of interactions between the agents, their environment and the organization. Hence, interactions are present at every level of reorganization support.

3.3.3.1 Organization Support for Reorganization

The organization support consists of a specification of the reorganization process and responsibilities.

In Hübner, J. F. and Sichman, J. S. and Boissier, O. (2004) and Sorici et al. (2012), the reorganization process is specified by a dedicated organizational specification (see 3.8). Specific roles are defined to (i) analyze the needs for reorganization, (ii) design organizational adaptations, (iii) implement the reorganization, and (iv) archive the organization's evolutions.

Further more, the reorganization process is clearly detailed in a social plan. Figure 3.6b gives a representation of such a plan. First, the organization must be (i) *monitored* in order to detect failures. Using the



(a) The reorganization group (b) The reorganization scheme

Figure 3.6: Organizational Specifications for Reorganization using the $\mathcal{M}$OISE framework (Sorici et al., 2012)

detected faults, the agents have to (ii) *design* one or several candidate adaptations, based on the past experiences or by expert analysis. Then, one of the reorganization is (iii) *selected*, before being (iv) *implemented* by modifying the OS.

On the one hand, such an explicit reorganization design coordinates and guides the agents through the process of reorganization. On the other hand, it also enables the agents to identify the roles and the constraints of the reorganization.

And so, the agents can control and reorganize the OE level of the reorganization process, by recruiting new organizational designers, for example.

3.3.3.2 Agent support for Reorganization

The previous section focused on the collective support of reorganization. We now study how the agents can carry out the reorganization process.

For example, in the TROPOS4AS framework (Morandini et al., 2009) the agents react to NCS and apply the corresponding recovery plan by either tuning their own behavior, reorganizing their relationships, or creating/deleting a new agent (see Section 3.3.2.1).

Nevertheless, the agents do not reason about the collective rules. In fact, such a micro-macro loop is set-up at the design phase. The TROPOS part brings traceability of requirements through the design phase until run-time. This way, the emergence is restrained into fixed solution space. Nevertheless, the high-level requirements cannot evolve at run-time.

In Bora and Dikenelli (2009) the agents have clear comprehension of the concept of fault-tolerant role (OS level knowledge) in order to manage the number of replica of the role (OE level reorganization). The agents can evaluate dynamically the actual criticallity of their role. When an agent detects a critical role of his, it calculates the number of replicas needed to ensure the fault tolerance, depending on the number of available resources.

In Oren et al. (2007) the agents use an argumentation framework, to manage the contract enforcement in a MAS. Such a negotiation is performed to decide how to relax and/or enforce the rewards and sanctions.

The framework could, potentially, support an OS level reorganization, but it is not used in such a manner. A possible extension of such a framework would be to negotiate a new contract (OS level) when too many relaxations have been done.

To do so, the agent must be aware of the OS, understand it and reason adequately.

For example in Glaser and Morignot (1997), the agents have representation of the roles proposed by the organization and those they expect to adopt. They can analyze the difference between the two roles and define a new role as a compromise.

The same for Hübner, J. F. and Sichman, J. S. and Boissier, O. (2004) which use expert agents (so-called designer and selector) in order to redefine the OS at run-time. Several designer agents propose a specific configuration each –roles, cardinality, plans and norms– and calculate the corresponding plan of changes. Then, these plans are submitted it to a selector agent which choose the new organization to set up.

In all of these frameworks, the reorganization is supported by direct interactions between the agents using the environment as a communication channel.

3.3.3.3 Environment Support for Reorganization

The environment dimension allows the agents to interact and share information and resources. Therefore, it can be used to help the agents in the reorganization process.

The Trace&Trigger adaptation framework (Alberola et al., 2012) propose a specific infrastructure for handling the reorganization in the environment.

First, it offers a publish/subscribe mechanism –so-called *Event Tracing*– for agents to be aware of the organizational adaptations at run-time. The Event Tracing module publishes the events if and only if some agents have subscribed to them, and so, it reduces the traffic generated by the organizational data, which are delivered to the organization manager agents only.

Then, a *Reorganization Facilitator* service helps the organizational agents to calculate and evaluate the possible changes to the organization and their cost.

Finally, an *Organization Management* allows the agents to submit organizational changes and provide the management of open and dynamic multi-agent organizations. Such a module manages OE level reorganization operations: dynamic entrance and departure of agents in the organization, agents' services' information modification, relationships and role update.

The ORA4MAS framework (Hübner et al., 2010) also provide an environmental support for OE reorganization. It uses artifacts to give the agent a representation of the organization state and to manage the OE level changes.

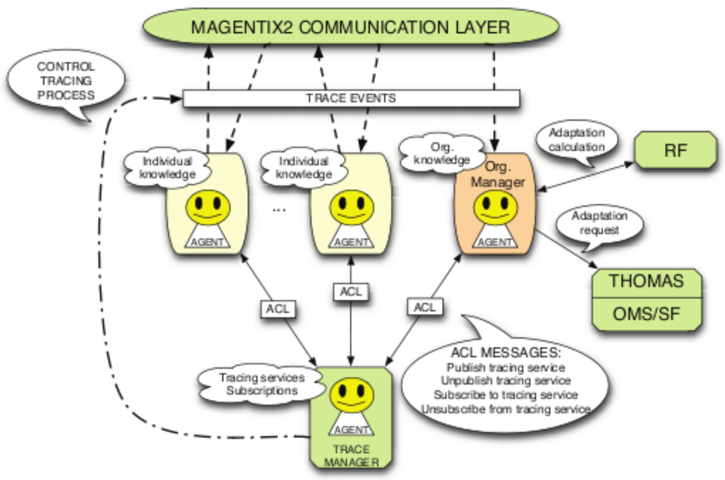

MAGENTIX2 COMMUNICATION LAYER
CONTROL TRACING PROCESS
TRACE EVENTS
Individual knowledge
Individual knowledge
Org. knowledge
Org. Manager
Adaptation calculation
RF
Adaptation request
THOMAS
OMS/SF
AGENT
ACL
ACL MESSAGES: Publish tracing service Unpublish tracing service Subscribe to tracing service Unsubscribe from tracing service
Tracing services Subscriptions
TRACE MANAGER

Figure 3.7: View of the Trace&Trigger's support for MAS Reorganization (Alberola et al., 2012)

The *OrgBoard* artifact represents the whole OS and OE, composed of one or several *GroupBoard*, *SchemeBoard* and *NormativeBoard* artifacts. A *GroupBoard* artifact allows the agents to adopt and leave a role in a group. It is linked to one or several *SchemeBoard* artifacts corresponding to the schemes the group is responsible for. A *SchemeBoard* artifact contains the plans, goals and missions of the scheme and use a *NormativeBoard* artifact to manage the mission commitments.

Sorici et al. (2012) propose an extension to the ORA4MAS framework to support OS level reorganizations. A dedicated *ReorgBoard* artifact allows the agents to (i) interrupt an OE, (ii) redefine the OS and (iii) restart the OE.

In particular, the design operations (e.g. *addRoleObligation*, *changeRoleCardinality*, *leaveRole*...) help the agents to design the reorganization plan, step by step, at the OS and OE levels. That is the missions and roles to leave before the reorganization, the changes on the OS and the new commitments after the reorganization.

3.3.4 Reorganization Issues

The two previous section studied how MAS reorganization can be designed and implemented. Nevertheless, reorganization raises several issues.

First, giving too much autonomy to the agents can lead to emergent behavior. And such an emergence is hardly controllable (Capera et al., 2003). Even when the degree of liberty of the agents is limited, such as in Morandini et al. (2009), the emergent behavior is still difficult to validate. And so, test and simulations are often required to check the result of such a behavior.

Further more, even planned reorganizations are not trivial. For example, Bora and Dikenelli (2009) point out some questions to answer when increasing the cardinality of a role. In fact, when duplicating a role the

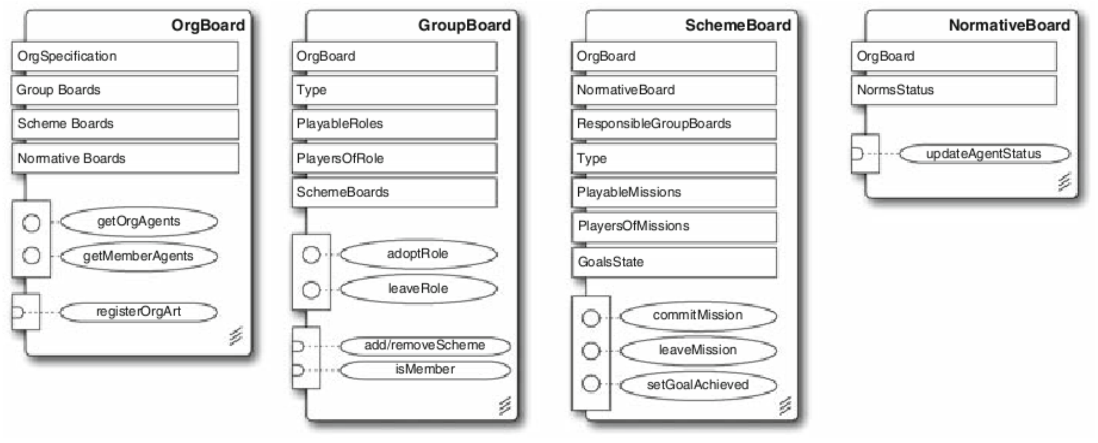


Figure 3.8: Representation of the Organizational Artifact in the ORA4MAS framework

designer or the agents should define if it should be adopted by the replicated agent or a different kind of agent. In the first case, it raises the problem of duplicating the other roles, and the second case raise the problem of the compatibility with the new agent's other roles.

Finally, the consistency of the MAS during the reorganization is not always guaranteed. Sorici et al. (2012) point out the problem of redefining the organization at run-time. The authors propose to stop the organization's and to restart it after the OS redefinition. Nevertheless it is necessary to define what to do with the running processes when stopping them ? While some processes can interrupted and, potentially, rolled-back, others are critical and should not be interrupted but waited for them to be ended.

3.4 Synthesis on Multi-Agent System for Governance

In this chapter, we have studied MAS from the governance perspective. We now synthesizes the main points of our review.

In MAOP, the MAS can be defined along the four VOWELS orthogonal dimensions: (i) the *Agent* dimension containing reactive and proactive programs (agents), (ii) the *Environment* dimension abstracting the resources and passive entities to be manipulated by the agents and mediating the interactions of the agents, (iii) the *Interaction* dimension enabling the different components of the MAS to communicate with each other, and (iv) the *Organization* dimension representing the collective behavior of the MAS –i.e. groups, roles, collective goals and plans, norms.

Our study of reorganization in MAS shows it is an adaptation process resulting from the contradictions between the objectives of the system and the conditions –or the constraints– in which the system is running. We can distinguish the Organizational Entity (OE) level reorganization from the Organizational Specification (OS) level.

Thus, reorganization can be represented by a micro-macro loop in which (i) the business processes can be adapted to fit (or improve) the objectives (OE level), or (ii) the objectives can be refined to suit to new external constraints or to suit to new business objectives (OS level). Nevertheless, some mechanisms should ensure a certain inertia in the MAS, preventing it to reorganize all the time, and never perform its main function.

In this chapter, we have given the pieces that convince us MAS suit naturally to the governance tasks. The different MAS dimensions can be used to model and implement the different governance concerns, identified in Chapter 1.

- The *governance strategy* can be defined as a MAS organization specifying the global objectives to be fulfilled by the governed system.
- The *governance tactic* can be enacted by the agents reasoning about the objectives and defining the ways to control the Machine-to-Machine (M2M) infrastructure.
- The *governance policy* can be ensured by environmental artifacts, abstracting the M2M infrastructure to the agents and controlling its components according to the rules defined by the agents.
- Finally, the *governance refinement* can be performed through a reorganization process.

In the next chapter, we highlight the main points suited in our state-of-the-art review. We synthesize our conclusions on the M2M governance issues and how the literature provides tracks to solve them. This will the base for our proposition of an M2M governance system.

CHAPTER 4

State of the Art synthesis

Contents

The first part of our thesis was dedicated to the study of the state of the art with respect to our objective: a governance framework for M2M systems. In this chapter, we synthesize the key elements responding to our needs for such a governance.

First, Section 4.1 addresses the issues for governing M2M systems in comparison with classical Information Technology (IT) and Service Oriented Architecture (SOA) systems. Then, Section 4.2 review how the governance can be carried on by software systems to provide more automation. Then, Section 4.3 shows how a MAS can be used to implement such an automated governance.

4.1 What governance for the M2M ?

M2M systems are faced to several proper issues which are introduce by the nature of such an infrastructure. We consider an N-tier architecture in which (i) M2M applications share (ii) an M2M Core Platform and (iii) network infrastructures (including M2M gateways and repeater nodes) to access to (iv) M2M devices (wireless sensors and actuators). We can identify three issues which are specific to such an N-tier M2M architecture.

Constrained Infrastructure. The first original constraint of such an infrastructure is the nature of the M2M devices. On the one hand, the main benefit of such devices is their very low price. On the other hand, human intervention to install and maintain them is expensive.

In addition, M2M are very limited in terms of computational power, network communications and power supply. Hence, it is important to guarantee their lifetime.

Horizontal Deployment. In order to reduce the deployment and maintenance cost of M2M infrastructures, an N-tier architecture allows to share them beyond all the stakeholders in a so-called “horizontal” deployment. That is, each stakeholder participates in the infrastructure for its competences only and share it with the other stakeholders.

Nevertheless, such an horizontal deployment requires the stakeholders to be guaranteed that their constraints and requirements would be fulfilled.

Scalability. As the M2M infrastructure is open and shared by all the stakeholders, we are faced to the scalability issue. This is enforced by economic trends that envision billions of M2M devices to be connected in a short future. In turn, this will potentially increase the number of M2M services, and so, the variety of requirements.

4.1.1 Horizontal Governance of M2M

Several works contribute to these technical issues, at the Wireless Sensor and Actuator Network (WSAN) level (Akyildiz et al., 2002; Erdene-Ochir et al., 2011; Lampin et al., 2012) or the platform level (Baugé and Bernat, 2008; Lam et al., 2009; Ückelmann et al., 2010; Foschini et al., 2011; Ückelmann et al., 2011).

Nevertheless, the experience of the SensCity project illustrate the problem of the lack of clear governance of an horizontal deployment. That is, an M2M governance should guarantee the alignment of the stakeholders’ individual business requirements and the technical constraints of the M2M infrastructure. And so, we can identify two dimensions in the M2M governance:

Vertical requirements for each M2M application.

Horizontal requirements to guarantee the whole infrastructure's functioning.

On the one hand, most of the works here address the *vertical* dimension Wefer, Gerold (2009); Spiess et al. (2009). On the other hand, the European Telecommunications Standards Institute Technical Committee on M2M (ETSI TC M2M) proposes a reference architecture for horizontal M2M infrastructures (ETSI, 2011).

In the next section, we synthesize our study of how the governance of IT and SOA systems is enacted by governance frameworks and tools.

4.1.2 The Governance Cycle

As M2M technologies and projects are still at early stage, it is not possible to compile enough experience feedback to define a set of best practices, as for IT and SOA governance.

Governance has been used for IT and SOA systems for many years now. It is possible to derive from these models, governance principles to apply to the M2M context.

Our review of the Control Objectives for Information and related Technology (CobiT) governance (Hayes et al., 2007) and IBM's SOA governance (Brown and Cantor, 2006) frameworks leads us to decline the governance at along levels:

The Governance Strategy corresponding to the *IT Goals Definition* (CobiT) or to the *Governance Requirements Planning* (SOA),

The Governance Tactic corresponding to the *Process Goals Definition* (CobiT) or the *Governance Approach Definition* (SOA),

The Governance Policy corresponding to the *Activity Goals Definition* (CobiT) or the *Governance Model Enabling* (SOA).

The Governance Refinement process evaluates the performance of the system in order to improve and reorganize it and refine the governance. This step corresponds to the *Governance Measurement and Realignment* (CobiT) or the *Governance Measurement and Refinement* (SOA).

Moreover, a *Declarative Approach*, such as `WSLA` (Ludwig et al., 2003), makes the governance objectives and performance metrics explicit for the system. Therefore, it is possible automate the monitoring and the evaluation the system's performance regarding the governance objectives. We can distinguished these objectives between *functional* and *non functional requirements*.

Functional Requirements refer to the objectives the system should achieve in terms of business logic –i.e. the services delivered.

Non Functional Requirements refer to the objectives the system should achieve in terms of performance –i.e. how the services are delivered.

4.2 Automating the M2M Governance

Our choice for an automated governance is motivated by the nature of M2M systems and is confirmed by the high proportion of errors due to human intervention (Andrzejak et al., 2002).

4.2.1 Endogenous versus Exogenous control

Our study of adaptive systems lead us to classify such systems into two categories: (i) endogenous adaptive systems, and (ii) exogenous controllers.

Endogenous adaptive systems are systems that run both the business logic process and a control and adaptation process.

Simple approaches exhibit good performances such as stochastic or probabilistic algorithms (Andrzejak et al., 2002; Erdene-Ochir et al., 2011), swarm intelligence (Schoonderwoerd et al., 1997) and clustering algorithms. These approaches are lightweight and are very reactive. Nevertheless, all of these approaches have a limited requirement expressivity and reasoning capability.

Agent-based systems and MAS use more intelligence in order to deal with the complexity of the system (Jennings, 2001). Software agents are capable of reasoning about their environment in order to decide the best action to perform to reach their goal Andrzejak et al. (2002); Capera et al. (2003); Carr et al. (2009). The governance objectives and decisions can be explicitly programmed by such a MAS (Kota et al., 2012). Yet, such a reasoning can cause computational and network overheads.

Exogenous controllers separate the business logic process and the governance process, so they can be distributed, and so, save resources.

Middlewares aim at abstracting the deployment layer, so that the system can focus on the business logic. Arregui et al. (2001) and Kessis et al. (2009) clearly illustrate that separation of concerns. Middleware technologies enable to control external systems without interfering in the business logic. Nonetheless, if they allow adaptation policies to be set up to control the legacy system, they do not focus on the management reasoning.

Autonomic Computing is a paradigm which aims at fulfilling such a gap. The autonomic reference model, as defined by IBM (2006), is composed of *Autonomic Elements* implemented by five complementary modules, so-called the MAPE-K architecture. Such an element participates in a layered architecture that provide as many degrees of abstraction. Further more, Weyns et al. (2010) propose a decentralized model that allows more flexibility and more reactivity.

Such autonomic models provide a mean to enact the governance policy as defined by the IT staff. Nevertheless, in both case, they do not reason about the expressed policy, as noticed by Maurel (2010). Such a reasoning is typically addressed by the MAS community, as discussed in the next section.

4.2.2 MAS-based Control

MAS typically use a middleware approach (e.g. the AgentScape middleware) in order to abstract the managed resources, so that the agents can focus on the governance concerns (Overeinder et al., 2002; Kousaridas et al., 2010; Oyenan et al., 2010).

Furthermore, MAS are especially suitable to deal with open and dynamic systems. For example, Aldewereld et al. (2010) support the design, the deployment and the maintenance of such systems. It combines the ALIVE middleware to abstract the deployed resources with the OperA framework (Dignum, 2004) to represent the business objectives and the stakeholders.

The tools and paradigms provided by MAS makes it a promising technology for support a full governance process. In the following section, we focus on the MAS definition and approaches which contribute to the governance perspective.

4.3 Multi-Agent Systems to carry the Governance

According to Boissier et al. (2011), Multi-Agent Oriented Programming (MAOP) is the synergistic combination of four dimensions of MAS programming. Demazeau (1995) named these four dimensions as the VOWELS model.

In this section, we summarize how these dimensions can be used to implement the different steps of the governance cycle.

The Organization dimension is concerned with the collective behavior of the MAS. Such an organization can be build either with (i) a *bottom-up* approach, or (ii) in a *top-down* manner. The latter approach suit better to our purpose, as it allows to define explicitly the system's requirement in a declarative manner, providing a clearer separation of concerns (Brazier et al., 2010; Bresciani et al., 2004; Chan and Sterling, 2003; Dignum, 2004; Herssens et al., 2010; Hübner and Boissier, 2002; Pitt et al., 2006; Sycara et al., 2009).

In fact, the organization's roles, goals and norms provide an high-level abstraction of the system which is understandable by both a human and an agent, just as `WSLA` for SOA. Then, following the norms, the agents can take the appropriate decisions to govern the system.

In particular, the $\mathcal{M}$OISE framework Hübner and Boissier (2002) decompose the organization into two orthogonal dimensions –structural and functional– bounded together by a normative dimension. This allows us to clearly separate the functional and non-functional requirements of the governance framework. Further more, Hübner, J. F. and Sichman, J. S. and Boissier, O. (2004) and Sorici et al. (2012) show that it is possible to reorganize these dimensions to refine the organization.

Therefore, it suits to the definition of high level objectives, that is the *governance strategy*.

The Agent dimension represents the active part of the MAS. An agent is a program than enables reactive and proactive processes and interacts with the environment and the other agents in a collective behavior. Agents are used to reason about the system and focus on the decision process to adapt it (Kota et al., 2012; Alberola et al., 2012; Brazier et al., 2010; Sycara et al., 2009; Picard et al., 2009).

In addition, the agents can reason about their organization. They can detect contradictions between the global objectives and their individual goals (Glaser and Morignot, 1997), or with the environment's state (Picard and Gleizes, 2004). Based on this analysis, they can determine whether to redefine the Organizational Specifications (OS) Bora and Dikenelli (2009); Glaser and Morignot (1997); Hübner, J. F. and Sichman, J. S. and Boissier, O. (2004) –for formal and long term reorganization– or the Organizational Entity (OE) Hübner, J. F. and Sichman, J. S. and Boissier, O. (2004); Morandini et al. (2009) –informal and punctual adaptation. But they can also stabilized the system by evaluating the cost of such a reorganization (Alberola et al., 2012; Kota et al., 2012)

Hence, agents can be used to decide the appropriate governance actions to perform, that is, the *governance tactic* for applying and refining the strategy.

The Environment dimension provides an abstraction layer to access to the resources (Weyns et al., 2007). That is, the environment makes possible to separate the business and the governance concerns (Sycara et al., 2009; Picard and Gleizes, 2004).

For example, Dragone et al. (2011) propose an hybrid component- (business logic) and agent-based (adaptation logic) approach to clearly separate the two concerns.

The Agents & Artifacts (A&A) meta-model (Omicini et al., 2008) adds another degree of abstraction. In fact the business processes' control can be defined by artifacts. This way the artifact can provide to the agents a comprehensive view of the system and high-level operations to control the governed system.

Hence, only the processes which require adaptability are implemented by the agents. The passive entities are located in the environment and controlled by the agents, reducing so, the overhead cost generated by the decision process.

Further more, the environment can be used to provide a support to facilitate the reorganization process at both the OS (Sorici et al., 2012) and the OE Alberola et al. (2012) levels.

Thanks to these properties, environmental artifacts can embody the *governance policy* that is applied to the governed system.

The Interaction dimension. It enables the different entities of the MAS to communicate with each other (Bellifemine et al., 2008). Interactions appear to be the support for coordination and adaptation in the MAS.

Therefore, the *governance refinement* will be supported by the MAS' interactions. That is, following our governance purposes identified previously, (i) the organization drives the agents' behavior to satisfy high level governance objectives, (ii) agents interact with each other to coordinate themselves and determine the governance tactics to adopt, (iii) agents use artifacts to observe and control the governed system, (iv) artifact interact with each other to decompose the monitoring and control procedures in order to (v) enact the governance policies on the governed system, (vi) agents analyze the performance of the governance based on the artifacts' measurements and notifications, and (vii) refine the governance objectives by adapting the organization.

Using these four dimensions, we can clearly separate the different governance concerns in the MAS. Such a separation of concerns, couple with the decentralized nature of MAS, will help to deal with the complexity of the M2M governance.

4.4 Transition

In our state-of-the-art chapters, we studied and reviewed several work contributing to the problem of a governance for M2M systems. Chapter 1 addressed the governance principal and studied how applicable they were to the M2M issues. Then Chapter 2, reviewed the different approaches for controlling and managing a system, according to governance specifications. Finally, in Chapter 3 we have show how MAS can be used for expressing the different governance concerns.

As a conclusion of this first part, we can identify the following points to remember for our M2M governance proposal.

1. M2M systems have specific concerns which should be explicitly addressed by the governance: (i) the N-tier architecture involving multiple stakeholders, (ii) the horizontal and vertical requirements, (iii) the M2M devices' constraints (low energy, low capacity, low network connection and long lifetime expected), and (iv) the scalability management.

2. An M2M governance framework should include four different parts: (i) the *governance strategy* should express the global objectives to be ensured by the M2M infrastructure, (ii) the *governance tactic* should analyze these objectives and define the way the infrastructure should be managed, (iii) the *governance policy* should monitor and control the M2M infrastructure according to management rules, and (iv) the *governance refinement* is the process through which the governance is analyzed and redefined at all of the three levels to improve the governance performance.

3. The specificity of the M2M governance concerns make it necessary to automate the governance process. Regarding the constraint of the resources an the openness of such infrastructures an exogenous governance system should be privileged.

4. MAOP offers an appropriate paradigm to implement such an M2M governance framework: (i) the different MAS dimensions suits to the three layers of the governance framework (the organization for the strategy, the agent level for the tactic and the environment for the policy), and (ii) the governance refinement can be performed through a MAS reorganization process.

These different aspects are discussed in the next part of this thesis, in which we introduce our proposal for an adaptive governance of M2M systems and its implementation, the AGaMeMnon framework.

Part II

Proposal for an M2M Governance

CHAPTER 5

A Multi-Agent Oriented Programming based Machine-to-Machine Governance

Contents

In Part I, we have identified several issues and requirements regarding Machine-to-Machine (M2M) governance. This part describes our proposal based on the previous part's conclusions.

That is, M2M governance should balance horizontal and vertical requirements, taking into account specific constraints for each stakeholders of the N-tier architecture. We also stated that such an M2M governance should be specified by a governance strategy and applied following a governance tactic as a decision support for governance policies that monitor and control the M2M infrastructure. Further more, M2M governance should be continuously refined in order to adapt to the M2M infrastructure's dynamics and scalability issues.

In addition, our literature review revealed that such an M2M governance could be automated using a Multi-Agent System (MAS). We showed that an Multi-Agent Oriented Programming (MAOP) approach would allow us to separate the different governance concerns along the different MAS dimensions.

In this part, we propose a *Multi-Agent Oriented Programming approach for a decentralized and distributed adaptive governance for Machine-to-Machine systems*.

This chapter introduces our M2M governance framework called AGaMeMon (*Adaptive Governance MAS for M2M*). Each section of this chapter outlines a layer of our M2M governance proposal that will be detailed in the three next chapters. We focus on the design concepts of the M2M infrastructure MAS governance. Implementation details, using the JaCaMo MAOP framework, and its integration within the SensCity M2M infrastructure will be discussed further in Part III.

We use the organization dimension to define the M2M governance strategy (Section 5.1). Then, the agent dimension embed the main reasoning processes supporting the M2M governance tactics and refinement decisions (Section 5.2). Finally, the environment dimension provide an abstraction of the M2M infrastructure by the means of artifacts which embed the M2M governance policies (Section 5.3).

Figure 5.1 provides an overview of the AGaMeMon governance for the SensCity M2M infrastructure. Each applicative "vertical" requirement is translated into a "vertical" Moise organizational specification while the "horizontal" specifications is defined from the European Telecommunications Standards Institute Technical Committee on M2M (ETSI TC M2M) recommendation.

The governance agents commit to the roles corresponding to the responsibility they take. According to their responsibility, the agents decide whether to follow this strategy definition or not and how to apply it or adapt it.

To do so, they use artifacts that encapsulate the different components of the SensCity infrastructure and provide different policies to control them.

In the remainder of this dissertation, *M2M entity* refers to a part of the M2M infrastructure which is governed by the governance layer. Such M2M entities can be a *client M2M application*, an *M2M core platform* server, an *M2M gateway* of a *group of M2M devices*. A group of M2M devices is referenced by a single identifier and aggregate a consistent set of M2M devices providing the same service, provided by the same stakeholder and located in the same geographical area.

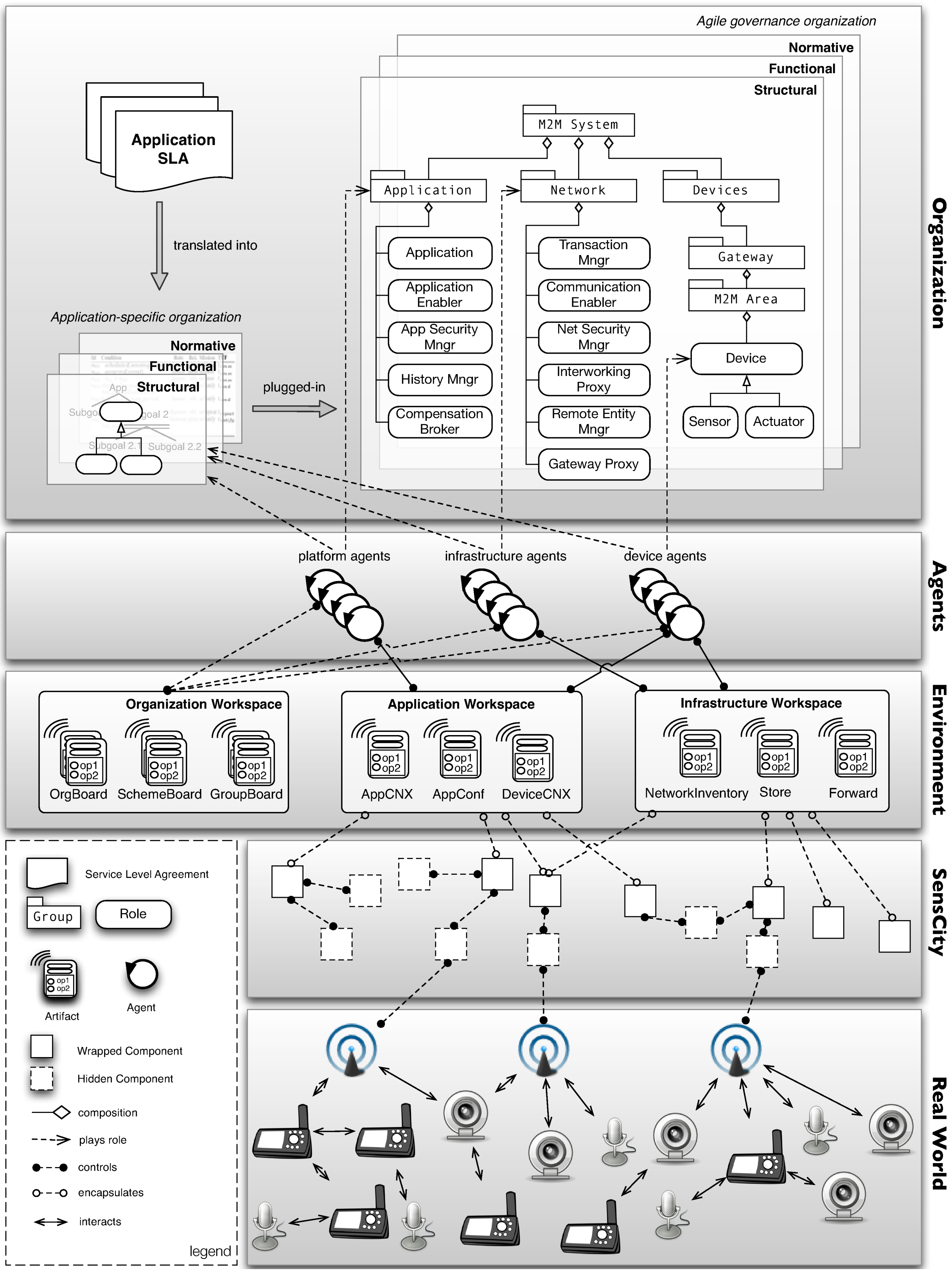


Figure 5.1: AGAMEMON governance of the SensCity M2M infrastructure.

5.1 Organization based Governance Strategy

The organization level defines the global objectives of the MAS corresponding to the governance strategy. That is, it defines the expected behavior of the M2M infrastructure.

Such an Organizational Specification (OS) can be understood by the agents, so they can govern the M2M infrastructure based on it. Furthermore, they can reason about organization to choose whether to follow it or not and then to adapt it to the situation.

In order to clearly separate the *vertical* and *horizontal* concerns, the OS is divided into a set of several organization. In addition, this reduces the complexity of each organization. And so, it simplifies the declaration and the adaptation of each one.

The roles defined in these organizations corresponds to parts of the M2M infrastructure to govern. Social schemes define requirements for each organization as goals. When committing to such roles, the agents hold the responsibility for its good functioning. While adopting roles in the horizontal organization and one or several vertical ones, the agents binds the horizontal and vertical requirements.

Following, we describe the horizontal strategy OS (Section 5.1.1) and the vertical strategy OS (Section 5.1.2) represented by Figure 5.2.

5.1.1 Horizontal Strategy Organizational Specification

To define the horizontal strategy, we choose to follow the ETSI TC M2M standard recommendations (ETSI, 2011). As a matter of fact, to our knowledge, this work is the most advanced effort for building horizontal M2M infrastructure.

The ETSI TC M2M defines capabilities M2M entities should support for enabling interoperable M2M infrastructures. Each capability comes with a set of functionality.

Structural Specification. In order to make the agents aware of such capabilities and functionalities, we define an OS based on the ETSI TC M2M specifications, as shown in Figure 5.2 (b).

The root role *M2MMnger* is an abstract role which allows us to define governance constraints for the agents. That is, we define how the agents can be organized for governing the M2M infrastructure. All roles are compatible with each other, so a single agent can assume the governance responsibility for all the capabilities. Further more, all the roles have a communication link, which means all the agents can communicate with each other.

The 11 sub-roles correspond to the different M2M capabilities defined by the ETSI TC M2M. The roles' maximal cardinality is infinite so the number of agents playing each role can grow if needed. The minimal cardinality is set to 1 if the capability is mandatory and 0 if it is optional.

Functional Specification. For each capability, we define a social scheme that contains the goals corresponding to the its functionalities.

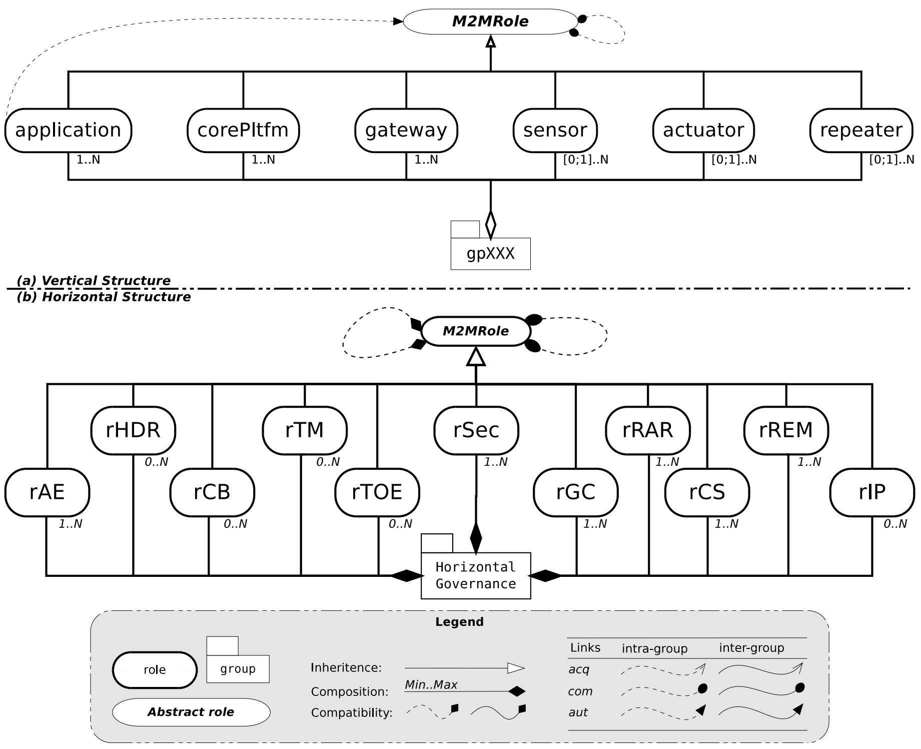


Figure 5.2: Structural Specifications for (a) a vertical M2M contract and (b) the M2M Horizontal Strategy.

Each functionality is turned into a goal which should be maintained in parallel with the others. Some functionalities include sub-functionalities (e.g. *gREMFault*), and so we decompose the goal into sub-goals. Each goal is associated with one and only one mission, so each goal can be distributed to different agents.

Such a goal affectation is done by the means of deontic norms. That is, we use norms to permit and/or oblige the agents to commit to the missions. The deontic relation –i.e. permission/obligation– is determined by the optional or mandatory property of the functionality.

5.1.2 Vertical Strategy Organizational Specification

An horizontal M2M infrastructure enables interoperability between independent M2M entities deploy by different stakeholders. That is, virtual “vertical” –i.e. end-to-end– deployments are done dynamically.

Thus, it is necessary to define a contract between such stakeholders specifying the rights and duties of each. This specified as a Service Level Agreement (SLA) establishing which M2M are involved and the expected Quality of Service (QoS).

To do so, we defined a model of vertical M2M contract, issued from existing M2M SLA specified within the ARESA2 project (Barthel et al., 2010). Such a contract specifies (i) the M2M entities involved, (ii) the type of M2M services –i.e. *Data Collection* or *Command Sending*– and (iii) a set of QoS parameters.

Such a contract is then turned into an OS, so that the agents can understand and analyze it. To do so, we define a vertical OS template which can be tuned according to the terms of the contract.

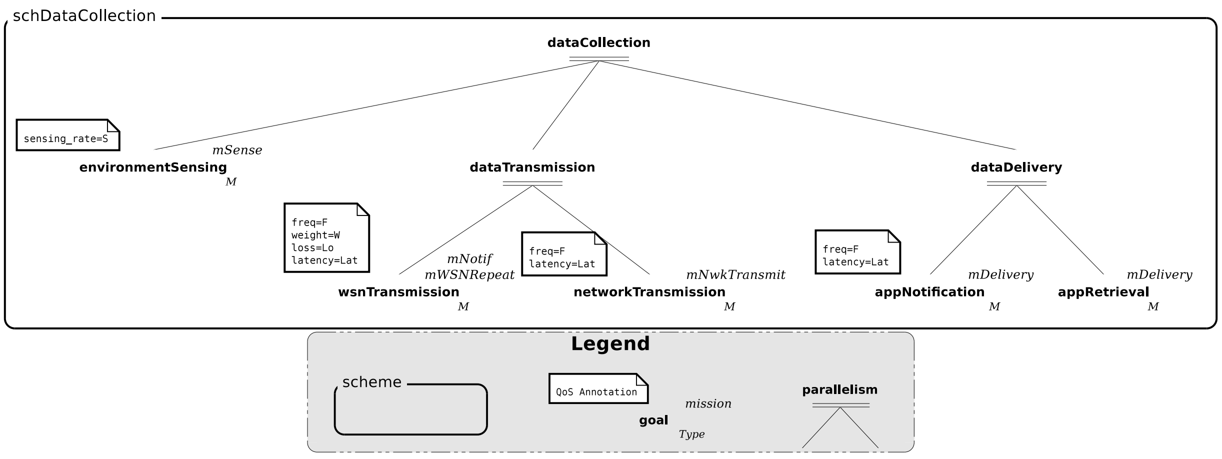


Figure 5.3: Graphical representation of the *Data Collection* Social Scheme.

Structural Specification. Roles in a vertical organization represents the different M2M entities involved into the contract, namely, (i) the client *application*, (ii) the M2M core *platform*, (iii) the M2M *gateways*, (iv) WSAN *repeaters* and (v) the applicative devices –i.e. *sensors* and *actuators*.

Figure 5.2 (a) describes such a vertical structure template, where the group gpXXX represents the contract `XXX`. It is composed of the roles involved in the corresponding contract. When governance agents adopt such roles they engage the M2M entity they govern in this contract.

Functional Specification. The terms of the contract are represented as a social scheme. We have defined two types of M2M SLA social schemes, namely, *schDataCollection* and *schCommandSending*, representing the two types of M2M services. We give an example of such scheme in Figure 5.3.

Such schemes are composed of parallel maintenance goals which represent requirements for each steps of the vertical M2M process. Goals are annotated so that the agents are aware of the QoS level for each requirement. Missions encapsulate such goals so that norms can assign the requirements to govern to the corresponding M2M entity governance agents.

Horizontal / Vertical Binding OS. As the OS is split into several organizations, we need to provide a means for coordinating such an horizontal/vertical requirement binding.

We define an horizontal scheme that commits agents in the horizontal organization to the vertical ones. Such a scheme, described in Figure 5.4, corresponds to the insertion of a new SLA proposal. Agents playing horizontal roles must (i) recruit agents for being responsible of one the M2M entities involved in the vertical contract, (ii) which should adopt the corresponding vertical roles if it evaluates the SLA feasible, and then (iii) configure the M2M infrastructure accordingly to deploy the SLA.

With such a scheme, we can bind some of the vertical requirements with the horizontal ones. Though, other horizontal requirements are dependent on vertical ones. We chose to define such a binding as a knowledge of the agents.

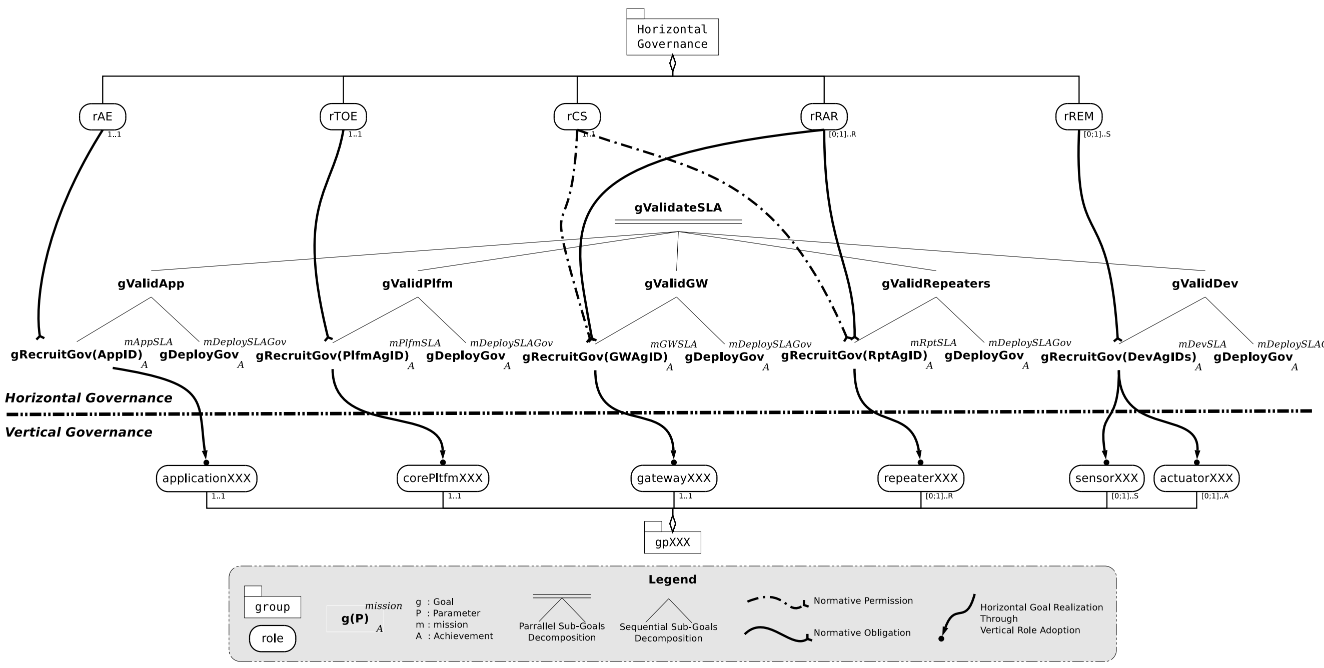


Figure 5.4: SLA Validation process and commitment.

5.2 Agent based Governance Tactics

Agents are the decision-making entities of the governance layer. Each agent govern at most one M2M entity of the infrastructure. They adopt one or several roles in the organization corresponding to the part of the governance for which they assume responsibility of. Hence the agent layer fulfills two governance objectives.

On the one hand, it *enacts* the governance by interpreting the organizational requirements and defining the actions to perform to manage the M2M infrastructure. Doing this, the agents bind the horizontal and vertical requirement through their role commitments.

On the other hand, it *refines* both the horizontal and the vertical governance. That is, the agents detect objectives that are impossible to fulfill or ways to achieve them more efficiently.

In this section, we give an overview of such a governance agent reasoning cycle (Section 5.2.1) and describe the three types of governance agents we have defined (Section 5.2.2).

5.2.1 Governance Reasoning Cycle

We define the governance tactic process based on our scalability governance definition (see Section 1.3.1.4 (p. 31). That is, the agents try to enforce the satisfaction of all the requirements. When it is impossible to satisfy all the requirements, they try to determine an acceptable trade-off to make the whole M2M infrastructure effective.

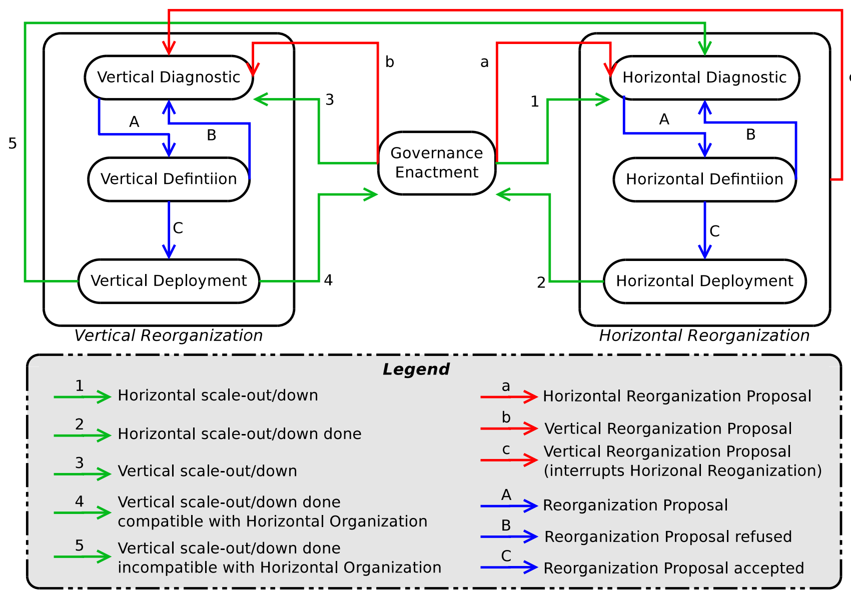

Vertical Diagnostic
Vertical Defintiion
Vertical Deployment
Vertical Reorganization
Governance Enactment
Horizontal Diagnostic
Horizontal Defintiion
Horizontal Deployment
Horizontal Reorganization
A
B
C
1
2
3
4
5
a
b
c
Legend
Horizontal scale-out/down
Horizontal scale-out/down done
Vertical scale-out/down
Vertical scale-out/down done compatible with Horizontal Organization
Vertical scale-out/down done incompatible with Horizontal Organization
Horizontal Reorganization Proposal
Vertical Reorganization Proposal
Vertical Reorganization Proposal (interrupts Horizonal Reoganization)
Reorganization Proposal
Reorganization Proposal refused
Reorganization Proposal accepted

Figure 5.5: Governance Agent's Reasoning Cycle.

Such an agent reasoning cycle can be divided into three parts: (i) *governance enactment*, (ii) *vertical reorganization* and (iii) *horizontal reorganization*, as depicted in Figure 5.5.

Governance Enactment. At first, the agents apply the requirements –i.e. organizational specifications. To do so, they configure the M2M infrastructure accordingly, using the governance artifacts.

When the M2M infrastructure fail to meet the requirements, they use a *scale-up* tactic. In a first step, they try to enforce the infrastructure behavior by the means of governance policies, embedded by the governance artifacts, which regulate the M2M entities activity.

If such a regulation still fails to meet the requirement, *scale-out* or *scale-down* tactics might be performed. To do so, the agents also need to analyze the strategy and to perform a reorganization.

Horizontal Reorganization. Horizontal reorganization consists of analyzing the horizontal strategy and adapt the organization to improve its performance. As the horizontal OS is specified from the ETSI TC M2M standard, we do not want the agents to modify the OS. Hence, horizontal reorganization only addresses the Organizational Entity (OE).

Such a reorganization can be triggered either (i) when M2M entities are deployed or undeployed (infrastructure scale-out/down), (ii) when an agent is overloaded (governance scale-out), and so, requests more agents to share its governance tasks, or (iii) when agents are overloaded (governance scale-down), and so, try to reduce the size of the governance layer –i.e. the number of agents.

This leads to the creation, enlargement or diminution of coalitions –i.e. informal organizations, which the agents are aware of– of agents sharing the same governance objective. We choose to use coalitions instead of structured organization, to clearly separate the governance concerns. In fact, such an organization comes under a tactic matter and not a strategy one.

Horizontal \ Vertical	*application*	*platform*	*gateway*	*device*
Application Enablement	*AppAg*			
Reachability, Addressing and Repository				*DevAg*
Communication Selection				*DevAg*
Remote Entity Management				*DevAg*
Transaction Management				*DevAg*
Generic Communication		*InfAg*	*InfAg*	*DevAg*
Telco Operator Exposure		*InfAg*		
Interworking Proxy		*InfAg*		
History and Data Retention		*InfAg*		
Security	*AppAg*	*InfAg*	*InfAg*	
Compensation and Brokerage	*AppAg*	*InfAg*	*InfAg*	*DevAg*

Table 5.1: Horizontal and Vertical requirements binding and the corresponding affectation to the agents

Vertical Reorganization. Vertical reorganization consist in analyzing and adapting the vertical strategy. Each vertical contract is treated independently, as it corresponds to a separate OS. Contrary to the horizontal reorganization, vertical reorganization is performed at the OS level.

Such a reorganization is triggered either (i) when new vertical contract are proposed (infrastructure scale-out) or (ii) terminated (infrastructure scale-down), or (iii) when requirements are no longer bearable for the M2M infrastructure (infrastructure scale-up).

5.2.2 Governance Agents Types

In the previous section, we have described the base behavior of the governance agents. We now specialize such behavior to take into account the M2M stakeholders' interests.

We can identify three main stakeholders involved in the M2M infrastructure: (i) the M2M application provider (*AppAg*), which develops value-added services, (ii) the M2M devices provider (*DevAg*), which sets up and maintains sensors and actuators, and (iii) the M2M infrastructure Telco (*InfAg*), which provides the intermediate M2M infrastructure –i.e. the M2M core platform, M2M gateways and WSAN repeaters. Therefore, we define three types of governance agents –namely *application*, *device* and *infrastructure* agents– which focus on each stakeholders point of view.

Each type of agent is involved in both the horizontal and the vertical governance, and so, they commit to roles in the different organizations. This allows the agents to bind the horizontal and the vertical requirements by the means of roles, as illustrated in Table 5.1.

Application Agents. Application agents represent the client M2M application point of view. That is, they govern the M2M infrastructure with respect to the application objectives. Hence, an *AppAg* agent tries (i) to *maximize the utility of M2M devices* the application subscribed to, whilst (ii) *minimizing the cost* induced by the subscriptions and the contract violation.

As specified in Table 5.1, an *AppAg* agent is involved in the *Application Enablement*, *Security* and *Compensation and Brokerage* horizontal roles, and participate as an *application* in the vertical organizations. As a matter of fact, it maintains the horizontal requirements by fulfilling all the vertical ones.

Application agents enact the governance by validating the application's subscription requests and configuring the M2M infrastructure accordingly. They regulate the application's activity by enforcing their subscription rights when to many violations occurs.

Reorganization is triggered when the governed M2M application requests a new subscription to M2M devices. This is performed in two manners. On the one hand, they create new vertical organization corresponding to the subscription and initiate the recruitment/validation process defined in Section 5.1.2. On the other hand, they create/enlarge a coalition dedicated to a specific M2M application governance, if they can not afford new vertical commitments.

Device Agents. Device agents represent the M2M device group point of view. That is they govern the M2M infrastructure regarding the M2M devices objectives: (i) to *maximize the number of client applications* it provides, whilst (ii) *maximizing the M2M devices lifetime*.

As specified in Table 5.1, a *DevAg* agent is involved in the *Generic Communication*, *Reachability, Addressing and Repository*, *Communication Selection*, *Remote Entity Management*, and *Compensation and Brokerage* horizontal roles, while being involved in the *device* roles in vertical organizations. Such as the application agent, it maintains the horizontal requirements by fulfilling all the vertical ones.

Device agents enact the governance by analyzing and validating the terms of subscription requests and configuring the M2M devices accordingly. They regulate the device activity by enforcing the devices commitment when failing, but also by updating routes to avoid network overload and maintain a high level of battery.

Reorganization is triggered when the governed M2M devices fail to fulfill their vertical requirements or when their lifetime become critical. This can lead to a vertical reorganization to define lower QoS attributes. But it can also require horizontal reorganizations. For example, to provide an alternative routes, other M2M devices could be hired as *repeaters* in the vertical organization. It is also possible to split a device group in order to separate good devices and critical ones. Then, a new vertical organization is created with lower SLA for the critical device group.

Infrastructure Agents. Infrastructure agents represent the M2M Telecommunication Operator (Telco) infrastructure point of view. That is they govern the M2M infrastructure regarding the Telco objectives: (i) to *ensure the end-to-end transmission* of M2M messages, whilst (ii) *minimizing the infrastructure cost*.

As specified in Table 5.1, an *InfAg* agent is involved in the *Generic Communication*, *Telco Exposure*, *History and Data Retention*, *Interworking Proxy*, *Security* and *Compensation and Brokerage* horizontal roles, and in the *platform* or *gateway* vertical roles depending on the type of M2M entity it governs. Contrary to the application and device agents, the infrastructure agent has an horizontal focus on the M2M infrastructure. Hence, it realizes the vertical requirements by ensuring the horizontal ones for each of them.

Infrastructure agents enact the governance ensuring the availability and functioning of the M2M infrastructure. They regulate the core platform and gateways by controlling their workload and balancing it between several servers.

Thus, infrastructure agents' reorganization is mainly horizontal. If all the agents responsible for the deployed M2M core platforms –or the M2M gateways– reject the SLA, the coordinator agent evaluates the possibility of deploying a new platform server –respectively, a new gateway. This leads to the instantiation of a new *InfAg* agent responsible for the newly deployed M2M entity.

5.3 Artifact based Governance Policies

The artifact level of governance provides a means to make the governance effective on the M2M infrastructure. Governance artifacts provide an abstraction the governed M2M entities.

Such an artifact monitors an M2M entity using infrastructure perception modules. Based on such percepts, it notifies the agents about the current state of the M2M infrastructure through governance properties, for performance statistics, and events, for critical states requiring a quick governance decision.

The agents can then control the M2M entity using governance operations. Such operations can trigger actions directly on the M2M infrastructure by the means of infrastructure controllers. But governance operations can perform more complex governance processes, such as set up and configure the monitoring components or the governance policies.

Governance artifacts embed governance policies which allow for an automatic governance control loop. A governance policy is a set of simple rules, stating how to act on the M2M infrastructure when some events happen.

We have defined three types of governance artifacts, the *Application*, the *Device* and the *Infrastructure* artifacts. Each type of artifact provides specific governance perception, events, properties, operations, controllers and policies to address the different M2M entities' governance concerns.

In this section, we first describe how we define governance policies as ECA rules, and then we give an overview of the governance artifact architecture, illustrated in Figure 5.6. We do not detail each type of governance artifact, in this chapter, as this is not relevant to our overview purpose.

5.3.1 Governance Policy

We define a governance policy as a *"parametrized reactive regulation behavior"* that enact a fast control loop on the M2M infrastructure. To define such a regulation, we found in Section 2.2.1.1 that a high-level declarative approach would provide more expressiveness.

Therefore, we choose to specify such a governance policy by the means of Event-Condition-Action (ECA) rules. An ECA rule defines *actions* to perform when an *event* occurs depending on a certain *condition*. Hence a governance policy can be defined as a set of ECA rules stating how to react to unwanted behaviors in order to enforce the requirements' fulfillment. Following, we describe the syntax for such ECA rules.

In such governance policy rule, an event E can be of three types: (i) M2M infrastructure percepts (`m2m(E)`), (ii) an event generated by another policy rule (`policy(E)`), or (iii) an event triggered by a governance tactic agent (`tactic(E)`). Then, conditions are defined a boolean operation holding on (i) the triggering event's properties (`E.p`), (ii) the policies' internal properties (`prop(P)`), (iii) the state of a requirement monitored locally (`req(R)`), or (iv) parameters set by the agents (`param(P)`). It is also possible

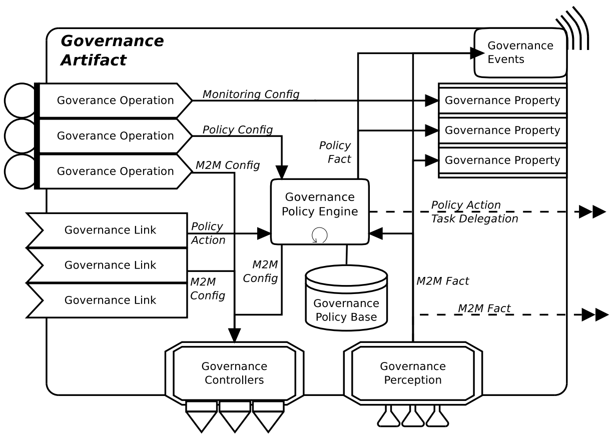


Figure 5.6: Architecture of a Governance Artifact.

to define rules (v) without a condition (`-`). Finally, the rules defines a sequences of actions to be performed, which can be (i) an alert to notify the agents (`alert(A)`), (ii) a policy internal action (`do(A)`), (iii) an action to be performed by another policy manager (`delegate(P,A)`), or (iv) an action to perform on the M2M infrastructure (`perform(A)`).

5.3.2 Governance Artifact Architecture

After having described the governance policies, we define in this section how such a governance policy process is implemented by governance artifacts. To do so, we have define an artifact architecture composed of different governance modules, as depicted in Figure 5.6.

Governance Properties. These are artifact observable properties, which allows the agents to monitor the state of the governed M2M entity. We can identify three types of governance properties: (i) M2M entity information, providing data about the M2M entity activity and the stakeholders' individual objectives, (ii) requirements fulfillment, which measures QoS realized by the M2M entity and compares it to the local requirements, and (iii) regulation policies performances, providing statistics about the governance policies triggered.

Governance Events. These are artifact signal sent to the agents to alert them about critical states requiring governance decisions. Such alerts can be either (i) M2M infrastructure events, notifying new/terminated registrations and requested subscriptions, (ii) requirement failures, providing requirement failures and violation details, or (iii) policy alerts that notifies about failing or underused policies.

Governance Operations. These are artifact operations the agents can use to act upon the governance artifact. Using such governance operations, the agents can (i) configure the M2M infrastructure, (ii) define local requirements to be monitored by the artifact, and (iii) activate, deactivate and tune the governance policies.

Governance Links. These are artifact linkable operations, which allow the artifact to be invoked other artifacts, provided that the agents have linked them together. Such links allow the artifact coordinate together by the means of different types of linked operations (dashed arrows in Figure 5.6): (i) M2M state propagation, (ii) policy coordination actions, and (iii) governance tasks delegation.

Governance Perception. These are M2M monitoring probes which feed the other governance modules. They monitor the M2M infrastructure in order to compute some `M2M events`. Then, depending on the nature of the event, it is dispatched to governance properties and governance events modules, but also to the governance policy engine. Such M2M events can also be forwarded to other artifacts using the governance links.

Governance Controllers. These are M2M infrastructure effectors, which allows the artifact to set up, (de)activate, configure and invoke the M2M entities. It provides several operators to the other artifact modules so they can control the governed M2M entity. Such controllers are used by (i) governance operations invoked by the agents, (ii) governance links invoked by the other artifacts, and by (iii) the governance policy engine to apply the policy actions.

Governance Policy Engine. It enacts the policies which have been activated by the agents. When events are triggered by the other artifact modules –i.e. governance operation, link and perception modules– it evaluates the policy rules to determine which actions to perform. Then, it performs the different policy actions specified by the selected policy(ies).

Such policy rules are stored in a *Governance Policy Base* which contains all the policies the artifact can run. Hence, the governance policy engine do not need to be specialized for each specific artifact. Further more, it is possible to define additional governance policies and add them to the policy base dynamically.

5.4 Transition

In this chapter, we provided an overview of our M2M governance framework proposal. Based on MAOP, we can define the whole governance process as a MAS. The governance strategy, tactic and policy are define along the organization, agent and environment dimensions, respectively, while the governance refinement is performed at each MAS level by the means of adaptation and reorganization interactions.

In the three next chapters, we detail each layer of the AGAMEMON M2M governance model: the organization based governance strategy in Chapter 6, the agent based governance tactic in Chapter 7 and the artifact based governance policy in Chapter 8. Then, we give more implementation details in Chapter 9 (Part III).

Chapter 6

Multi-Agent Organization-based Governance Strategy

Contents

In this chapter, we describe the first part of our proposal: the representation of the *governance strategy as a multi-agent organization*. As stated in Chapter 3, the organization in a MAS offers an adequate degree of abstraction so that the global objectives can be formulated and understood by both humans and software agents.

We have chosen the $\mathcal{M}$OISE framework to specify and implement the organization. As presented in Section 3.2.4, this framework allows to specify the organization along two orthogonal dimensions –structural and functional– which are joint using norms. Thus, it facilitates the specification of flexible organizations.

Furthermore, we have split the organizational specification into several ones. That is, the different strategy concerns represented by different organizations. We can clearly separate two part of such a strategy: (i) the horizontal strategy, and (ii) the vertical strategy.

This chapter details the different parts of the strategy definition. First, Section 6.1 sustains our choice to use several organizations. Next, we give the methodology for designing the horizontal specifications from the ETSI TC M2M's recommendations (Section 6.2). Then, we describe how vertical SLAs can be expressed as an Organizational Specification in Section 6.3. Finally, we specify how the horizontal and the vertical strategies are bound together in 6.4.

6.1 A Multi-Organizational Specification

In this section, we explain the reasons why we have chosen to split the governance strategy into several organizations. Afterwards, these different organizations will be detailed in Sections 6.2 and 6.3.

6.1.1 Horizontal vs. Vertical Strategy

Orthogonality of M2M Infrastructures. Our first motivation to split the strategy into several organizational specifications comes from the orthogonal conceptions of M2M systems. As described in Section 1.2.1.4 and 1.2.2.2, horizontal deployments is a willing trend for M2M infrastructure, but vertical deployments still offer better guarantees for ensuring the M2M requirements.

Our proposal provides a natural way to follow this trend by expressing the vertical requirements independently from the horizontal concerns.

First, an organization is specified in order to take into account the horizontal aspects of the M2M infrastructure. This strategy aims at enabling the infrastructure for all the stakeholders. Hence, it is agnostic to the applicative requirements, but focuses on the infrastructure's functioning. We follow the standard ETSI TC M2M functional specifications in order to define the governance strategy.

Then, the vertical strategy focuses on the applicative requirements –i.e. requirements for a client application to perform its services using the M2M devices. This strategy aims at managing contractual agreements between the different stakeholders. In order to do that, a vertical organizational template is defined, and instantiated for each "vertical".

Functional and Non Functional separation of concerns. Furthermore, distinguishing the horizontal and the vertical organizations matches the separation between the functional and non-functional requirements of the M2M core platform.

On the one hand, the horizontal concerns defined by the ETSI TC M2M (see Section 1.2.3.3) correspond to the functional requirements of the M2M: provide, enable and ensure an end-to-end communication between all the stakeholders. In fact, ETSI (2011) details the capabilities and functionalities the M2M platform should provide to be meet the ETSI TC M2M standard.

On the other hand, the vertical concerns specify how each vertical end-to-end communication should be ensured. That corresponds to the non-functional requirements of the M2M core platform. Though, each vertical specification should define both the functional and the non-functional requirements for each vertical contract.

6.1.2 Benefits of a Multi-Organizational Specification

While the differentiation of the horizontal and vertical concerns in the strategy definition has a "natural" motivation, splitting the strategy in such a manner offers two main advantages.

1. It clearly separates the strategy into orthogonal dimensions, as explained previously. This reduces the complexity of the specification, as each concern is clearly separated from the others. It also improves the flexibility of the strategy (re)definition. These two properties will simplify the agents' reasoning about the strategy and reorganization process.
2. It facilitates the intervention of the different stakeholders in the specification and redefinition of the objectives while guaranteeing the consistency and integrity of the overall organization.

In fact, when redefining the strategy it is necessary to stop the processes –at least those impacted by the redefinition– which were following it in order to keep the system consistent. Stopping a process could either mean to abort it, suspend it or even waiting for it to end depending on the critical nature of the process or of the reason to redefine the strategy.

Thereby, splitting the strategy into several ones will help to decide which organizational processes should be stopped or not. Though, more coordination from the agents is needed to perform such a strategic redefinition.

So, using separate and independent organizational specifications improves the quality and simplicity of the strategy definition and analysis thanks to a clearer separation of concerns and smaller specifications. In the next two sections, we describe the definition of the horizontal and the vertical dimensions of the governance strategy.

6.2 Organization for an Horizontal Strategy

In this section, we focus on the horizontal aspects of the governance strategy. Section 6.2.1 starts by describing why and how the ETSI TC M2M recommendation can be used as a basis for the definition of the horizontal strategy. Then, Section 6.2.2 and Section 6.2.3 detail the organizational specifications corresponding to this strategy.

6.2.1 Governance specifications from the ETSI TC M2M

As reported in Section 1.2.3.3, the ETSI TC M2M normalization group proposed a reference architecture (ETSI, 2011) for M2M systems (see also Appendix C).

That is why we use the ETSI TC M2M functional specifications to design the horizontal governance strategy.

In this specification, the M2M infrastructure is divided into three domains: (i) the Application Domain –out of the scope of the ETSI TC M2M–, (ii) the Network Domain, and (iii) the Device Domain .

The ETSI TC M2M defined a functional architecture, corresponding to the two latter domains, divided into three Service Capability Layers (SCLs). Each SCL is a set of 11 capabilities –given in Table 1.5– declined along the three following layers: (i) the Network layer (**N**) located in the Network Domain (the M2M Core Platform), (ii) the Gateway layer (**G**) located in both the Network and Device Domain, (iii) the Device Layer (**D**) located in the Device Domain. Then, each capability is characterized by a set of functionalities to be ensured in the corresponding layer.

We will focus on the Network Service Capability Layer (NSCL) to define the organizational specifications. Several considerations lead us to this choice:

- The core platform is deployed on broadband servers which allows us to deploy the governance layer either locally or remotely if needed (e.g. to save CPU or memory) considering only a minimal communication cost.

- The core platform is a central node of the M2M architecture which controls all of the other M2M entities remotely.

- However, it might also be a bottleneck in the infrastructure considering the scaling up of the infrastructure. So it will be necessary to monitor and adapt it to ensure the horizontal. Therefore, our governance layer will be *deployed on top of the M2M Core platform*.

Table 1.5 in Section 1.2.3.3 (p. 28) provides a list of such capabilities and their description. In Table 6.1, we give an example of three capabilities, and the corresponding functionalities, for the NSCL: (i) Application Enablement (AE), (ii) Telco Operator Exposure (TOE), and (iii) Remote Entity Management (REM). For readability purposes, sub-functionalities are separated by semicolons and we do not specify here whether the functionality is optional or not (see Appendix C for a full description of the NSCL functionalities).

Let us notice that the functionalities of these three capabilities addresses all the parts of the M2M infrastructure: AE's functionalities aims at serving the client M2M applications, TOE's functionality consists of managing the M2M Core Platform itself, and REM's functionalities are focused on the management of the M2M gateways and devices. Nevertheless, these functionalities do not address each part's requirements individually. It is an "horizontal" view of the infrastructure's requirements –i.e. "make it work for everybody".

Capability	*Functionality*
AE	– Expose: *Exposes the NSCL functionalities to the M2M applications through a single interface* – Register: *Allows M2M applications to register to the NSCL* – Routing: *Performs routing between the M2M applications and the appropriate NSCL capabilities ; Perform routing between the capabilities* – Record: *Generates charging records pertaining to the use of capabilities*
TOE	– Interworking: *Interworking and using of Core Network services exposed by the Network Operator [i.e. Interworking with other core platforms' services*]
REM	– Configuration: *Provides Configuration Management functions* – Performance: *Collects/Stores performance data* – Fault Management: *Collects/Stores faults data ; Provide Fault Management functions* – Connection: *Triggers connection establishment* – Upgrade: *Manages software/firmware upgrades ; Support Authentication and Authorization of a management authority ; Avoid conflicts upon multiple authorities (Applications, including Manufacturer) ; Validate upgrade integrity (if supported)* – Protocol Management: *Supports several management protocols ; Abstract management protocols through a common one* – Lifecycle Management: *Installs, Removes and Upgrades applications in M2M gateways and devices* – M2M Service Management: *Configures service capabilities in M2M gateways and devices* – M2M Area Network Management: *Manages the configuration of the mesh networks*

Table 6.1: Description of the functionalities* for three NSCL capabilities: *Application Enablement(AE)*, *Telecommunication Operator Enablement(TOE)* and *Remote Entity Management (REM)*.
(* Sub-functionalities are separated by semicolons)

Based on these specifications, we can build a multi-agent organization so that the agents can be aware of the governance strategy. The following sections describes the steps to build such an organization using the $\mathcal{M}$OISE framework.

6.2.2 M2M Capabilities: Horizontal Structural Specifications

The structural dimension of the $\mathcal{M}$OISE framework (see Section 3.2.4) consists of a specification of the system in terms of roles and relationship constraints between the roles. That is, the different capabilities the MAS exposes. Hence, we use this dimension to express the ETSI TC M2M's capabilities.

Figure 6.1 gives a representation of such a structural specification. It is composed of one root role – *M2MMngr*– and 11 sub-roles. The root role is an abstract role –that cannot be adopted by the agents– used

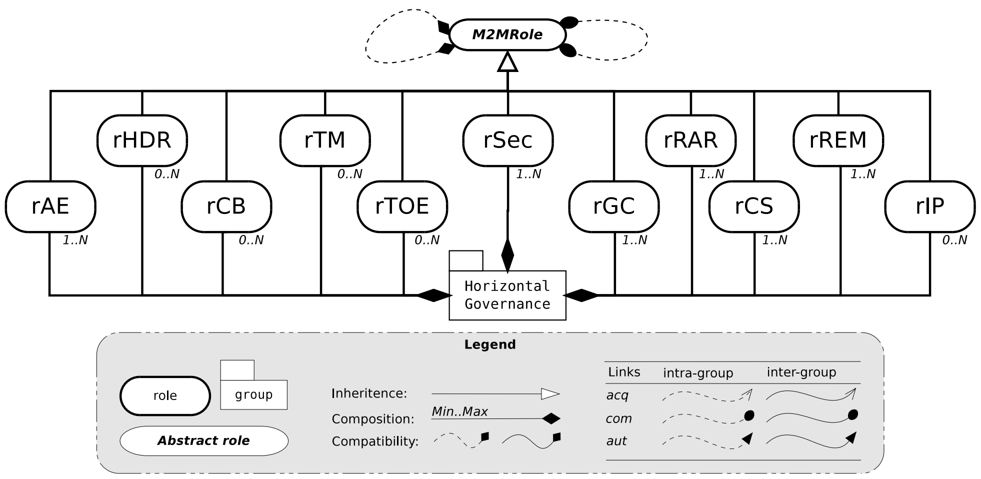


Figure 6.1: Structural Specification for the M2M Horizontal Strategy.

to specify the communication and compatibility constraints for the inherited sub-roles. Hence, all the roles are compatible –i.e. an agent can commit to all of the roles at once– and can communicate with each other.

Each capability of the NSCL is modeled as a sub-role of *M2MMngr* in the governance organization. Mandatory capabilities are represented by roles which have a minimal cardinality equals to 1 while optional ones' cardinality is 0. The maximal cardinality for all the roles is unbound which allows the agents to dynamically hire more agents to govern the corresponding capability, if needed. Table 6.2 gives a detailed description of the "horizontal" roles.

When the agents adopt such a role, they assume the responsibility to ensure that the M2M platform exposes correctly the corresponding capability of the NSCL. Each of these capabilities corresponds to a set of functionalities. In the following section, we describe how these functionalities can be turned into organizational specifications in $\mathcal{M}$OISE.

6.2.3 M2M Functionalities: Horizontal Functional Specifications

As the structural specification corresponds to the NSCL capabilities, the functional specification corresponds to the functionalities to be ensured by these different capabilities. In this section, we describe how we represent the NSCL functionalities in the organizational specifications.

The functional dimension of the $\mathcal{M}$OISE framework consists of a specification of the system in terms of social schemes (see Section 3.2.4). A social scheme is a tree decomposition of a goal into sub-goals using the SEQUENCE, CHOICE and PARALLEL operators. Such goals are grouped into missions. Such missions are assigned to the roles thanks to a norm triggered by a specific context.

Figure 6.2 gives a representation of such social scheme corresponding to the Application Enablement's functionalities. For each capability, we define a social scheme containing the goals corresponding to its functionalities. All the goals are maintenance goals that have to be fulfilled in parallel. Each goal is contained in

	Role	*Cardinality*	*M2M Capability*
Mandatory Capabilities/Roles	*rAE*	[1..N]	Application Enablement: *Single contact point to M2M applications*
	rGC	[1..N]	Generic Communication: *Single point of contact for communication with M2M gateways and devices*
	rRAR	[1..N]	Reachability, Addressing and Repository: *Manage groups of devices, routes and subscription to these groups*
	rCS	[1..N]	Communication Selection: *Manage and select alternative routes to the M2M devices*
	rREM	[1..N]	Remote Entity Management: *Manage the M2M devices' configuration, performance and failures*
	rSec	[1..N]	Security: *Authentication, Key hierarchy management, Integrity validation and verification*
Optional Capabilities/Roles	*rHDR*	[0..N]	History and Data Retention: *Archive and retain data from the devices*
	rTM	[0..N]	Transaction Management: *Manage atomic execution of complex operations on the devices*
	rCB	[0..N]	Compensation Broker: *Manage brokerage and compensation between M2M devices and M2M applications*
	rTOE	[0..N]	Telco Operator Exposure: *Manage interworking with intermediate Telcos services*
	rIP	[0..N]	Interworking Proxy: *Manage interworking with non-ETSI compliant systems*

Table 6.2: Correspondence between the Horizontal Roles and the ETSI TC M2M NSCL capabilities.

a single mission grouping, in turn, a single goal. Sub-functionalities are turned into sub-goals.

The cardinality of missions, respectively goals, correspond to the number of agents required to commit to the mission, respectively to fulfill the goal. As for the roles, an optional functionality is given a minimal cardinality of 0 and 1 for a mandatory one. The maximal cardinality is unbound so that number of agents can be raised at run-time to fulfill the goal.

Finally, the missions are assigned to the agents using norms. A norm expresses a deontic relation – permission or obligation– between a role and a mission. It is triggered when a certain condition is matched. Then agents can or must commit to the mission and fulfill the corresponding goals.

In our horizontal strategy, norms simply assign the mission to the role without any particular condition, as shown in Listing 6.1. The nature of the deontic relation is determined by the optional or mandatory criterion of the corresponding functionality.

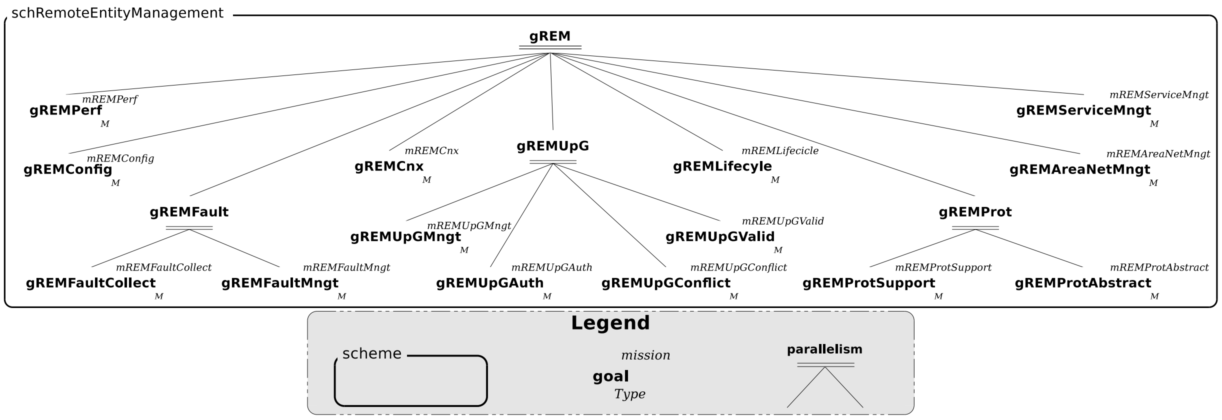


Figure 6.2: Functional Specification for the Remote Entity Management's functionalities.

```
<scheme id="schTOE">
 <goal id="gTOEInterW" type="maintenance"  min="0" />
</scheme>
<mission id="mTOEInterW" min="0" />
 <goal id="gTOEInterW"/>
</mission>
[...]
<norm id="nTOEInterworking" type="permission" role="rTOE" mission="mTOEInterW" />
```

Listing 6.1: Functional and Normative specification for the *rTOE* role, using the $\mathcal{M}$OISE XML specifications.

6.3 Organization Template for a Vertical Strategy

The organization presented in Section 6.2 allows to formalize the governance strategy for the horizontal concerns of the M2M infrastructure. As we noticed in Section 1.2.2.2, these NSCL capabilities and functionalities address all the infrastructure's functioning in a "make it work for everybody" spirit. They do not express each stakeholders' individual requirements.

Though, these vertical requirements must be taken into account implicitly by the horizontal ones. That is, horizontal requirements imply that all individual requirements must be satisfied.

For example, the REM's functionality "Fault Management" means all M2M devices' failures should be managed and balanced which implies for the governance system to manage each device's failures.

This section addresses such a vertical organizational design for the governance strategy. The link between the horizontal and vertical requirements will be discussed further in Section 6.4.

In this section, we aim at defining governance objectives for a vertical M2M application. This is achieved by an organizational representation of the SLA corresponding to a vertical "contract" encompassing the responsibilities of every involved stakeholders.

Devices	*Sensing Rate*	*Message Frequency*	*Message Size*	*Max Loss*	*Max Latency*
`Gas Meters`	4/H	1/Day	100B	$O(10^{-2})$	300s
	4/H	4/Hour	10B	$O(10^{-2})$	300s
`Gas Switch`	–	1/Week max.	10B	$O(10^{-3})$	30s
	–	1/Year exp.	10B	$O(10^{-3})$	30s

Table 6.3: SLAs for a gas management application. (From Barthel et al. (2010))

We first analyze the content of such contract in Section 6.3.1, before detailing its representation as an organizational template in Section 6.3.3 and Section 6.3.3.

6.3.1 M2M Vertical Contract

An M2M vertical contract engages the different entities of the M2M infrastructure necessary to perform an end-to-end vertical process –i.e. command to devices or data collection– namely, an M2M application and a set of M2M devices, but also the M2M core platform, the M2M gateway(s), and the necessary Wireless Sensor and Actuator Network (WSAN) repeaters.

The SLA defines the terms of the contract. That is, the tasks the M2M entities have the right or the duty to perform and the constraints to perform them. For each task, the SLA specifies (i) the condition in which the task can/must be performed, (ii) which entity should perform it, and (iii) the QoS Non Functional Requirements (NFRs) to perform it (e.g. frequency, latency or load). The tasks may correspond to actions to perform in the vertical process, but it can include other actions such as financial compensation, for example.

Barthel et al. (2010) give us some examples of typical SLAs for typical urban M2M applications. For instance, Table 6.3 shows the needs for a gas management application which uses (i) `Gas Meters` devices to monitor the clients' consumption, and (ii) `Gas Switch` devices to have an hand in the gas inlets.

- It needs to collect measurements from `Gas Meters` devices from once a day to four times in an hour. These measurements are to be done every 15 minutes. The message's size should not exceed 100 Bytes, if it compiles the whole day, or 10 Bytes, if the alert is reported directly. The message transmission should be less than 5 minutes and 1 % loss.
- In case, the application have to control the gas inlets (e.g. to prevent gas leakage), it can send a message to the `Gas Switch` devices that should be transmitted in less than 30 seconds, with less than 0.1 % loss. This message should not be bigger than 10 Bytes and not be sent more than once a week to each device, once a year in the usual case. In this case, no response message is required.

Such an example gives us the main elements to be defined in the vertical SLAs: (i) the direction of the communication (i.e. *data collection* or *command to devices*), (ii) the tolerated latency and (iii) loss of messages, (iv) the expected load on the infrastructure (i.e. the messages' *size* and *frequency*), and (v) the *sensing* (or sampling) rate –for data collection– or (vi) an expectation of *response* –for command sending.

Such an information gives an important clue about the expected load for the overall M2M infrastructure, particularly for the M2M devices, that the execution of such a vertical contract will cause.

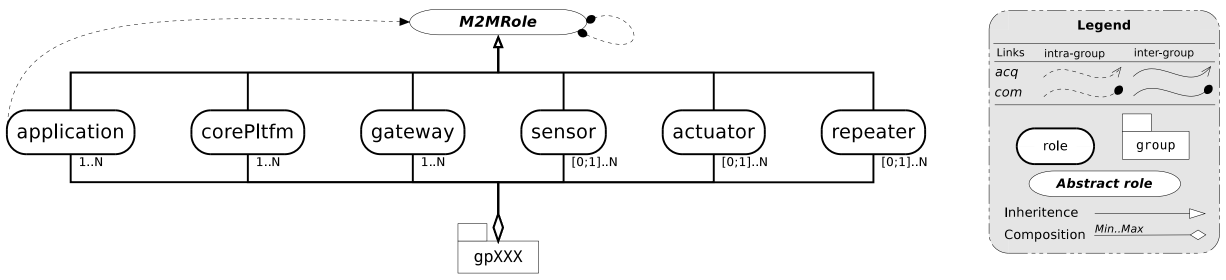


Figure 6.3: Structural Specification for an M2M Vertical Contract.

In addition, this contract should also include (i) a list of the M2M entities engaged in the contract, (ii) the time period delimiting the contract, and (iii) the financial compensation for the contract fulfillment and possible violations. Definition 4 provides a formal description of the elements contained in a vertical M2M contract in our governance system.

Definition 4: Vertical M2M Contract

Let be $C : \langle \mathcal{E}, P, \Sigma, f_c \rangle$ a vertical M2M contract, where

- *$\mathcal{E}$ is the list of the M2M entities' identifiers involved;*
- *P is the time period for the contract validity;*
- *$\Sigma : \langle \sigma_0, \ldots, \sigma_N \rangle$ is the list of the SLAs, where each SLA $\sigma = \{DC, CS\}$ can be either a Data Collection (DC) or a Command Sending (CS) requirement definition:*
  – $DC : \langle S, M, La, Lo, W \rangle$ denotes a Data Collection with a sensing rate of S, a message frequency of M, a tolerated latency of La, Lo loss, and for messages weighting W;
  – $CS : \langle R, C, La, Lo, W_C, W_R \rangle$ denotes a Command Sending with a boolean expectation of response R, a frequency of commands C, a tolerated latency of La and Lo tolerated loss, and for commands weighting W_C and responses weighting W_R;
- *f_c is a compensation function, denoted by:*

$$f_c : \mathcal{S}^2 \times \mathcal{H}_v \longrightarrow \mathbb{R} \tag{6.1}$$

where $\mathcal{S}$ is the set of stakholders and $\mathcal{H}_v$ is a violation history. That is, $f_c(s_1, s_2, h_v) = c$ assigns stakeholder s_1 to pay a compensation fee of c to stakeholder s_2, given the contract violations h_v.

6.3.2 Vertical Structural Specification Template

In this section we describe how we represent an M2M vertical contract in our governance strategy framework, before Section 6.3.3 describes the definition of the terms of the contract.

In order to represent the different parts of the system involved in the vertical contract, we use the structural dimension of the $\mathcal{M}$OISE framework.

We have defined a template that can be instantiated for each vertical contract. Figure 6.3 gives a representation of such a vertical structural specification. It is composed of a group (gpXXX) representing the contract XXX. Each M2M entity $e \in \mathcal{E}$ of the contract is represented by a role. Following is a description of each of these roles.

The *application* role represents the external client application. An agent adopting this role is responsible for ensuring the client application correctly receives the messages as expected and for controlling the commands sent to the devices do respect the SLA.

The *corePltm* role represents the M2M core platform. It is responsible for the platform to authenticate and delivers the messages and commands correctly. So, it makes sure the platform will not suffer overloads for this contract.

The *gateway* role represents a M2M gateway involved. As the *corePltm* role, it is responsible for the gateway to authenticate and delivers the messages and commands correctly. For example, an M2M gateway can be configured to retain some messages and sending them at once.

The *sensor* and *actuator* roles represents the applicative devices involved in the contract. As the *application* role, they check the compliance with the SLA by controlling messages send by the devices and commands they receive.

The *repeater* role represents M2M devices only involved to forward the messages between the applicative devices and the gateway. It is responsible for evaluating the load on these repeaters and, potentially, define the best route for the messages. Let us note that an applicative device can be used as a repeater for other devices.

Such a structural specification makes the structure of a virtual vertical deployment explicit for the agents on top of the horizontal M2M deployment. It can be customized following the needs of the contract. For example, the *actuator* role can be omitted if no actuation is needed in the contract.

In next section, we describe how we can assign QoS constraints to such a vertical structure using a functional specification in $\mathcal{M}$OISE.

6.3.3 Terms of the Service Level Agreement: Vertical Functional Specification

In order to represent the terms of the contract –i.e. the SLA– we have defined three template schemes –*schDataCollection*, *schCommandToDevice* and *schCompensationPenalites*– using the $\mathcal{M}$OISE framework. Thus, a vertical organization is specified using one or several of these template schemes. Such schemes are parametrized according to the QoS values given by the SLA.

6.3.3.1 Data Collection Organization

As stated in Definition 4 a *Data Collection* SLA is denoted by a tuple $DC : \langle S, M, La, Lo, W\rangle$ where S is the sensing rate, M is the message frequency, La is the tolerated latency and Lo the tolerated loss, and W is the maximum weight of the messages. The terms of such an SLA are turned into organizational specifications.

Figure 6.4 provides a representation of a social scheme corresponding to the *Data Collection*. It decomposes the steps of the data collection process as a goal sequence. This template is parametrized by the terms of the DC tuple.

The root goal (*monitorDataCollection*) is satisfied by maintaining the following sub-goals in parallel:

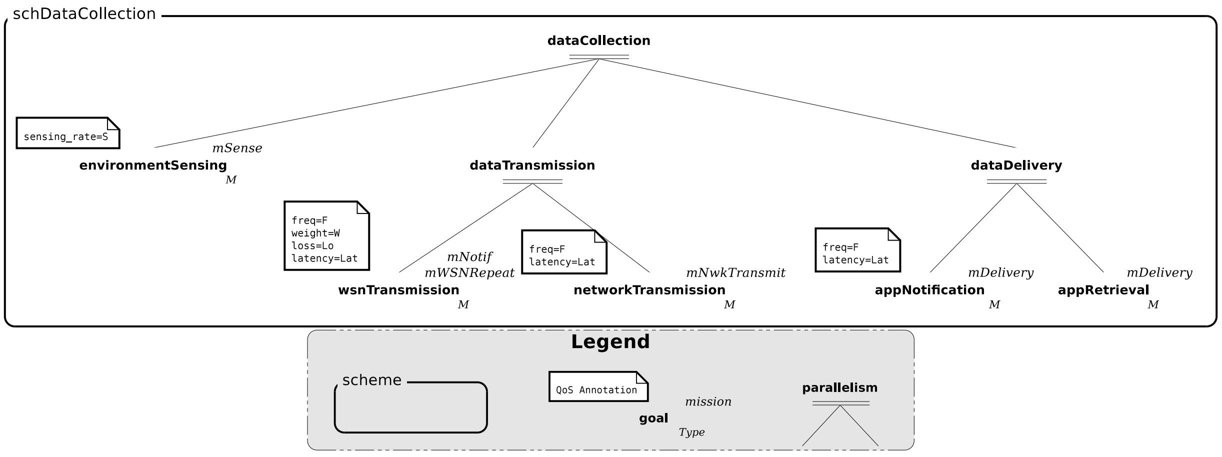


Figure 6.4: Graphical representation of the *Data Collection* Social Scheme.

1. *environmentSensing*, parametrized with `sensing_rate=S`, which consists of sensing the environment. This goal is contained by the *mSense* mission.

2. *dataTransmission* for notifying the sensors measurements. Notifications can be done either after each sensing or reporting several measure at once. Such a goal can be divided into the following sub-goals:

   (a) *wsnTransmission*, parametrized with the message frequency `freq=F`, where `F=ONEVENT` means a transmission for each measurement, the messages' weight `weight=W`, the tolerated loss `loss=Lo` and the tolerated latency `latency=Lat`. This goal is assigned to the agents playing the *sensor* and *repeater* roles by the missions *mNotif* and *mRepeat*, respectively.

   (b) *networkTransmission*, parametrized with the message frequency `message_freq=M` and the tolerated latency `latency=Lat`, which consists of data transmission to the broadband network. This goal is contained in the *mNwkTransmit* mission.

3. *dataDelivery*, which consists of delivering the sensor measures to the client applications. It is divided into the following sub-goals, contained in the mission *mDelivery*:

   (a) *appNotification*, which consists of notifying the client application of available messages, parametrized with the message frequency `message_freq=M` and the tolerated latency `latency=Lat`.

   (b) *appRetrieval*, which consists of delivering the pending messages when the client application requests to retrieve them.

Such goals are assign to the agents by the means of norms according to the role they play in the vertical organization, as shown in Listing 6.2. Each norm assigns a mission –i.e. a consistent set of goals– to a vertical role without any condition. That is, every agents must commit to their missions as soon as the social scheme is started.

```
<norm id="nDC0" type="obligation" role="sensor"      mission="mSense"   />
<norm id="nDC1" type="obligation" role="sensor"      mission="mNotif"   />
<norm id="nDC2" type="obligation" role="repeater"    mission="mWSNRepeat"   />
<norm id="nDC3" type="obligation" role="gateway"     mission="mNwkTransmit"   />
<norm id="nDC4" type="obligation" role="platform"    mission="mDelivery"   />
<norm id="nDC5" type="obligation" role="application" mission="mDelivery"   />
```

Listing 6.2: Norms assigning the vertical *Data Collection* goals to the vertical governance role, using the MOISE XML specifications

6.3.3.2 Command Sending Organization

As stated in Definition 4 a *Command Sending* SLA is denoted by a tuple $CS : \langle R, F, La, Lo, W_C, W_R \rangle$ where R is a boolean expectation of response, F is the frequency of commands the application has the right to send, La is the tolerated latency for a message to be transmitted, Lo is the tolerated loss, and where W_C and W_R are the maximum weight for, respectively, the commands and the responses.

As *Data Collection*, the terms of such an SLA are turned into a functional scheme specification. Figure 6.5 provides a graphical representation of such a vertical specification. Then, the scheme's social goals are parametrized using the CS tuple values.

The root goal (*cmdSentToDevices*) is satisfied by maintaining the following sub-goals in parallel:

1. *validateCmd*, which consists of validating the client application's commands before transmitting them to the devices. Such a validation is divided into two sub-goals, contained by the *mPlfmValid* mission:

    (a) *authCmdSender* to authenticate the client application which has sent the command.

    (b) *checkAppRights* to validate the client application rights with respect to the SLA. It is parametrized by the frequency of command `command_freq=F` the application can send.

2. *deliverCmd* to deliver the commands to the applicative devices so that they can perform them. Such a goal can be divided into the following sub-goals:

    (a) *cmdNwkDelivery*, parametrized with the tolerated latency `latency=La`, which consists of the command transmission through the broadband network, between the core platform and the gateway. This goal is contained by the *mNwkDelivery* mission.

    (b) *cmdWSNDelivery*, parametrized with the command messages' weight `weight=`W_C, the tolerated loss `loss=Lo` and the tolerated latency `latency=Lat`. This goal is contained by the *mCmdRepeat* mission.

    (c) *executeCmd*, which consists of executing the command by the applicative devices. It is contained by the *mDevCmd* mission. The goal is parametrized by a boolean value `response=R` stating whether a response is expected or not, and the maximum response message weight `resp_weight=`W_R.

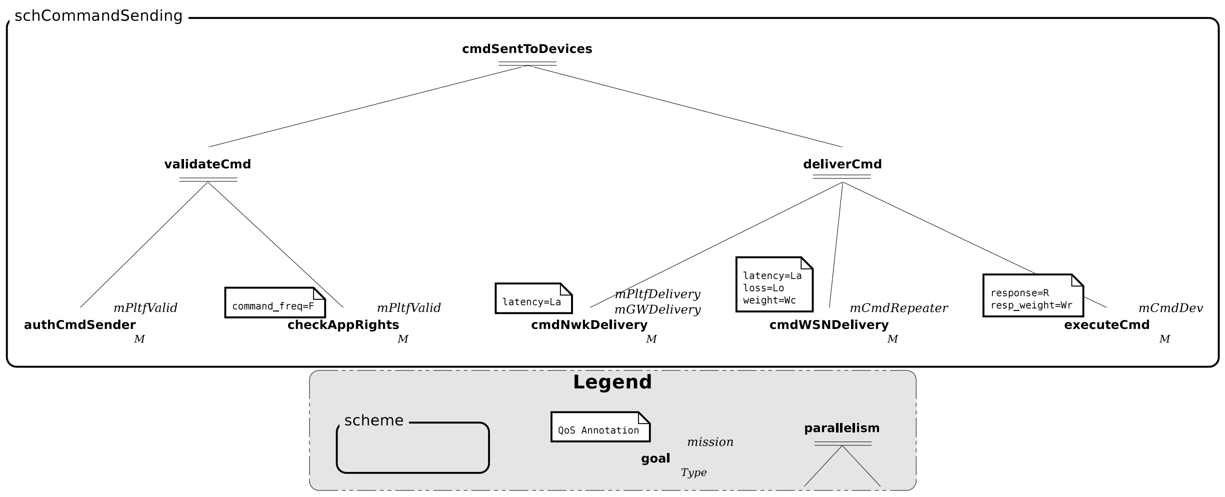


Figure 6.5: Graphical representation of the *Command Sending* Social Scheme.

```
<norm id="nCS0" type="obligation" role="application" mission="mValid" />
<norm id="nCS1" type="obligation" role="platform"    mission="mValid" />
<norm id="nCS2" type="obligation" role="platform"    mission="mNwkTransmit" />
<norm id="nCS3" type="obligation" role="gateway"     mission="mNwkTransmit" />
<norm id="nCS3" type="obligation" role="repeater"    mission="mWSNRepeat" />
<norm id="nCS4" type="obligation" role="sensor"      mission="mCmdExecute" />
```

Listing 6.3: Norms assigning the vertical *Command Sending* goals to the vertical governance role, using the $\mathcal{M}$OISE XML specifications

These governance goals are assigned by norms to the agents which are responsible for these M2M entities through the corresponding roles. Listing 6.3 describes such a normative specification. Once again, the role used in the norms depends on the ones defined responding the vertical contract needs. That is, the *mCmdExecute* mission can be assigned to *sensor* or a *actuator* role depending on the applicative needs.

6.3.3.3 Compensation and Penalties

Compensation and penalties address all the stakeholders involved in the vertical process realization: mainly the client application and the applicative devices owners, but also the Telco stakeholder providing the core platform, the M2M gateways and/or the infrastructure repeaters.

For clarity purpose, we aggregate the three latter stakeholder as one, the *infrastructure*. This choice seems reasonable to us, as they all are parts of the horizontal intermediate. Hence, in the SensCity project, they were all managed by the same stakeholder: Orange.

According to Equation (6.1) in Definition 4, the compensation strategy between two stakeholders $s1$ and $s2$ is defined by a function $f_c(s_1, s_2, h_v)$, where h is the contract violations history.

Such a strategy is implemented, in our governance framework, as a monitoring scheme. That is, norms

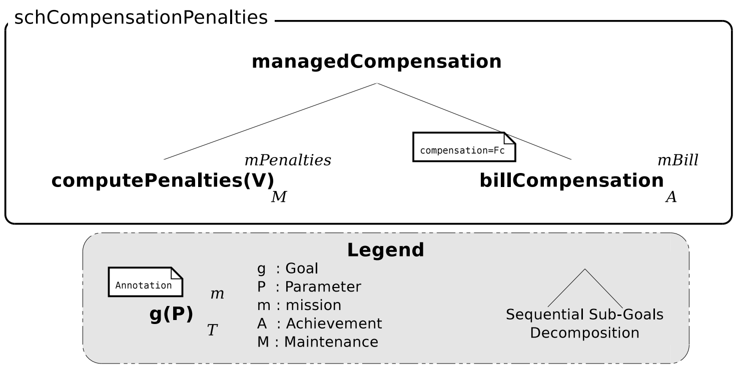


Figure 6.6: Graphical representation of the *Compensation and Penalties* Social Scheme.

```
<norm id="nCP0" type="obligation" role="all" mission="mPenalties"  />
<norm id="nCP1" type="obligation" role="all" mission="mBill" ttf="P.end"  />
```

Listing 6.4: Norms assigning the vertical *Command Sending* goals to the vertical governance role, using the $\mathcal{M}$OISE XML specifications

for this social schemes are triggered by the goal achievement or failure in the previously described social schemes.

Figure 6.6 describes such a social scheme and Listing 6.4, the corresponding norms. It is a simple sanction/reward process, excepted that the rewards are computed at the end of the contract, based on the contract violations.

First, all the agents are committed to the mission *mPenalties*, so that each SLA violation is reported to *computePenalties* goal's parameter `V`. In $\mathcal{M}$OISE, a goal parameter is a value to be set by the agents in order to fulfill the goal. That is, the agents will update such a value until the monitored scheme is running, as the *computePenalties* goal is a maintenance goal.

Then, compensations are computed based on the violations `V` reported previously, and the compensation function f_c given by the *billCompensation* goal's attribute `compensation=Fc`.

The corresponding norm (`nCP1`) has a time-to-fulfill value set to *P.end* corresponding to the end of the time period defined by the vertical contract (see Definition 4). In fact, billing the compensation fees means the end of the vertical contract. And so, agents participating in the actual vertical organization will be forced to put an end to the related M2M infrastructure activities and to compute the compensation bills.

6.4 Horizontal and Vertical Governance Binding: SLA validation

Before such vertical SLAs are managed by the governance framework, they must pass through an approval phase. That is, finding agents to validate the terms of the SLA and accepting the responsibility to ensure them. Hence, we formalize this process as a functional scheme.

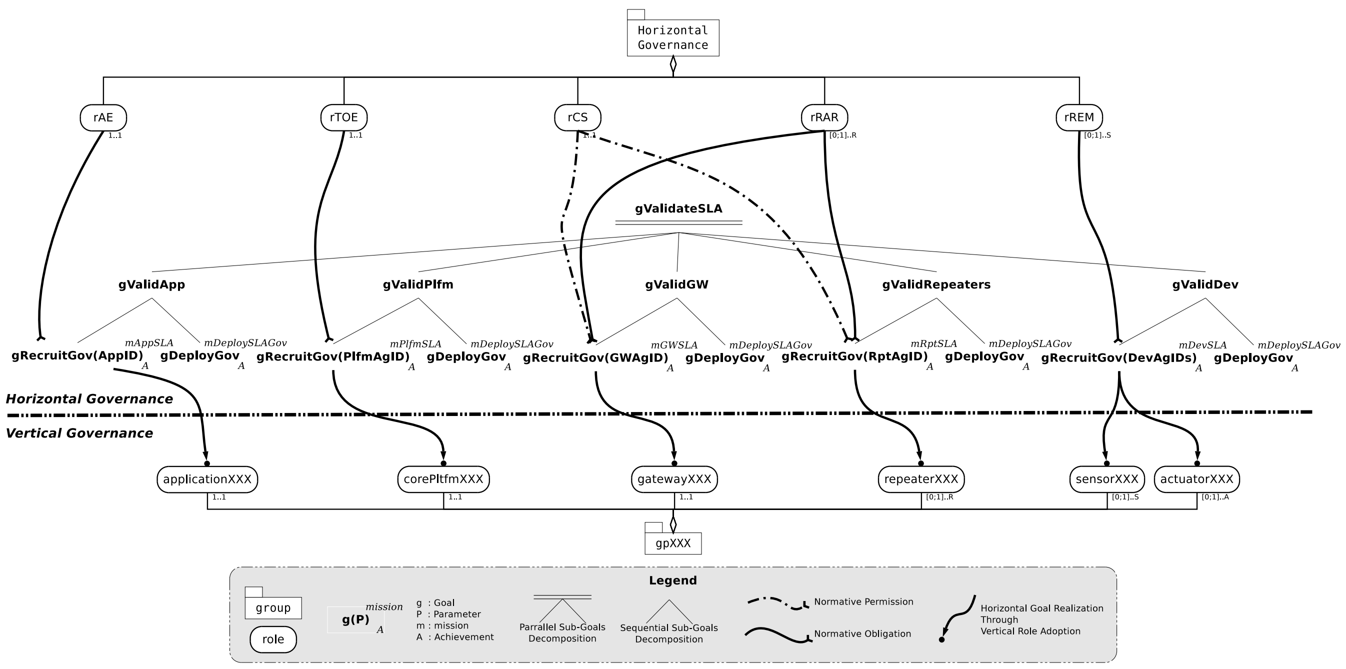


Figure 6.7: SLA Validation process and commitment.

The validation process is distributed over the different agents taking part in the horizontal governance, as described in Figure 6.7. The process can be divided into two phases: (i) the recruitment phase, and (ii) the deployment phase. This leads to the construction of a new instance of a vertical organization, as described in Section 6.3.3.

6.4.1 Recruitment phase

First, when an SLA proposal is received by the governance framework, the recruitment is initiated by the agent assuming the *rCB* role to fulfill its mission *mCBTokens*.

It consists of hiring other governance agents for validating the involvement of the M2M infrastructure's components. These components are represented by the list $\mathcal{S}$: $\langle App, PF, GW, RD, AD\rangle$ of M2M entities expected to participate in the contract, where App denotes the client application, PF the core platform, GW the gateway, RD the repeater(s), AD the applicative devices identifiers.

The SLA validation process is performed by the agents involved in the horizontal governance. Listing 6.5 shows how norms are used to oblige the horizontal governance agents to validate the different parts of the SLA, according to their horizontal responsibilities.

Each of these agents have to evaluate the feasibility of the requirements from a different point of view –i.e. its governed M2M entity's point of view. The goals *gRecruitGov* are achieved, when an agent have found reasonable to take the responsibility of the SLA. That is, the agent adopts the corresponding role in the vertical organization. Such a role adoption can be considered as a contract signature.

```
<norm id="nSLAValid0" type="obligation" role="rAE"  mission="mAppSLA"  />
<norm id="nSLAValid1" type="obligation" role="rTOE" mission="mPlfmSLA" />
<norm id="nSLAValid2" type="obligation" role="rRAR" mission="mGWSLA"   />
<norm id="nSLAValid3" type="obligation" role="rRAR" mission="mRptSLA"  />
<norm id="nSLAValid4" type="obligation" role="rREM" mission="mDevSLA"  />
<norm id="nSLAValid5" type="permission" role="rCS"  mission="mGWSLA"   />
<norm id="nSLAValid6" type="permission" role="rCS"  mission="mRptSLA"  />
<norm id="nSLADeploy" type="obligation" role="all"  mission="mDeploySLA"
  condition="fulfilled(mAppSLA) & fulfilled(mPlfmSLA) & fulfilled(mGWSLA)
             & fulfilled(mRptSLA) & fulfilled(mDevSLA)"          />
```

Listing 6.5: Norms assigning the vertical SLA validation to the horizontal governance role, using the $\mathcal{M}$OISE XML specifications

When all these goals are achieved and all the vertical roles assigned, meaning the whole SLA has been approved, the deployment phase is triggered by the norm `nSLADeploy`.

6.4.2 Deployment Phase

When all the parts of the vertical SLA are validated by the agents, the vertical governance strategy is deploy in three steps: (i) the instantiation of the governance strategy, (ii) the configuration of the infrastructure with respect to the SLA, (iii) the monitoring and management of the vertical SLA. All these agents participating in the vertical group are committed to the deployment of the SLA (mission *mDeploy*).
(i) The corresponding vertical organization is instantiated by the agents which have validated the SLA and adopted the corresponding vertical role in the group. The vertical functional scheme (*Data Collection / Command Sending*) corresponding to the terms of such an SLA contract can be started.
(ii) Thus, the agents are engaged to make the different parts of the N-tier infrastructure comply with the SLA. To do so, they have to perform the appropriate configuration so that the M2M entities will behave accordingly.
(iii) Then, the SLA management and monitoring strategy is governed by such a social scheme. The SLA compensations and penalties are managed through a monitoring scheme (*Compensation And Penalties*).

6.4.3 Multi-Organization Management

Thanks to such a multi-organizational design, each part of the governance strategy can be defined and maintained more easily. In fact, each organization is composed of a single group and have a limited number of roles, social schemes and norms. Thus, the overall organization specification's complexity is reduced as every organization is independent from each other.

Nevertheless, we can notice from Sections 6.4.1 and 6.4.1 that horizontal and vertical concerns are not completely independent. That is, some horizontal goals must be fulfilled by realizing vertical ones.

In addition, some vertical roles are impacted by the horizontal nature of the infrastructure. That is clearly the case for *platform gateway* or *repeater* roles, as they represent the horizontal infrastructure involvement in a vertical contract. But, it is also true for *device* roles: on the one hand, (i) an applicative device can be used as a repeater for other applicative devices, potentially using other vertical SLA and, on the other hand,

(ii) an applicative device can be used by different client applications with different NFR, and so different SLA.

Multi-Organization Issues. Such problems address several issues concerning a multi-organizational design, for example: *how to guarantee the organizational consistency ? how to define extra-organization constraints ? how to validate goal dependencies among several organizations ?* or *how to coordinate several organizations ?*

Such questions are not an exhaustive list of theoretical issues to be addressed, but they point out the complexity indirectly induced by a multi-organizational design. In fact, if each individual organization's complexity is simpler and more maintainable, the overall organization's complexity may remain.

Though, from specification point of view, smaller organizations would simplify the governance strategy definition and refinement. Hence, such a design is of high interest with respect to the number of vertical contract scale-out.

Agent-based Knowledge and Coordination. Regarding the objectives of this thesis, we chose to define the multi-organizational constraint and coordination knowledge at the *governance tactic* level. That is, we give the agents the knowledge to coordinate the horizontal and vertical roles and goals.

Such a choice is strengthened by another concern: *governance refinement*. The governance refinement process consists of the adaptation of the governance strategy based on the measured performance of the system, as described in 1.1.3. That means the governance strategy should be able to evolve from the original specification.

As the agents implement governance tactics to fulfill strategic objectives, they are best situated to analyze the feasibility of such objectives taking into account the infrastructure constraints and the governance performance.

As stated in Hübner, J. F. and Sichman, J. S. and Boissier, O. (2004) and Sorici et al. (2012) these organizations can evolve along different dimensions: e.g. roles, cardinality, links, missions, norms, deadlines. . . This allow the governance strategy to suits to the infrastructure's dynamics and to the vertical requirements evolution. As an example, if more agents are needed to manage a part of the M2M system, the cardinality of the corresponding roles can be increased.

Such a reasoning process is typically an agent capacity. An agent is free to decide whether to follow or violate the norms. It can be regulated by reinforcement or punishment, but it also provides a way to detect irrelevant specifications: an agent might violate a norm because it is impossible to satisfy a goal.

In the next chapter, we address how the governance tactics can be defined and implemented by the agents. We describe how the agent can deploy the governance strategy, reason about it to adapt the different governance tactics and analyze the performance to refine the strategy.

Chapter 7

Agent-based Governance Tactic

Contents

The previous chapter described how the governance strategy can be defined using Organizational Specifications. This chapter's purpose is to define how the M2M governance tactic can be described and implemented using agents. This governance layer has two interests:

1. **Governance Enactment.** The governance tactic is a means to apply the strategy. That is, the tactic layer interprets the objectives defined at the upper level in order to define the action to perform to manage the governed system.

   Further more, the governance tactic layer allows to bind the horizontal and the vertical strategy definitions together. In fact, as we mentioned in Section 6.4.3, the horizontal strategy's requirements implies the satisfaction of the vertical ones. So agents have to take into account both the horizontal and vertical strategies to decide how to apply them.

2. **Governance Refinement.** As it interprets the strategy, the governance tactic can also reason about it. That is, it can detect objectives that are impossible to fulfill or ways to achieve them more efficiently.

   Such a governance refinement can be performed in both the horizontal and the vertical dimensions. The agents can refine these strategies through a multi-agent reorganization processes. While these processes have a certain degree of independence, they may also interfere due to the horizontal-vertical dependencies.

The governance tactic layer is applied by agents for the realization of these two objectives. The agents are the decision-making entities of the governance layer. A governance agent is specialized in a specific concern of the N-tier architecture. With this specialty, it is involved in the fulfillment of one or several requirements specified at the governance strategy level. They bind the horizontal and the vertical strategies together by adopting one or several roles in the different orthogonal organizations.

We have defined three types of agent, that all follow the same governance tactic reasoning process. Each type of agent is specialized in a specific M2M entity point of view: (i) application agents (*AppAg*), (ii) device agents (*DevAg*), and (iii) infrastructure agents (*InfAg*).

First, we describe, in Section 7.1, the base architecture of a governance agent. Such an architecture implements the basic governance tactic process. That is, (i) M2M infrastructure *regulation* according to the horizontal and vertical governance strategies, (ii) *vertical reorganization* to add, remove and refine vertical SLAs, and (iii) *horizontal reorganization* to react to M2M infrastructure deployments and to adapt the governance accordingly.

Then, in Sections 7.2, 7.3, and 7.4, we define the agents specialyzing the governance tactics, for each M2M entity. Thus, we need to define (i) how the agents bind the horizontal and the vertical strategy, (ii) what part of the M2M infrastructure do they need to control to fulfill the strategy objectives, and (iii) how they reason about the adaptation according to the entity they represent.

Finally, we conclude this chapter, in Section 7.5, by synthesizing our contribution and defining the requirements for the lower level layer of our framework, based on the agent's abstraction needs.

7.1 Base Architecture of Governance Tactic Agents

In this section, we define the agents' tactics for applying the governance strategy, for reasoning about its efficacy and efficiency, and to come to adaptation decisions. This is performed through two types of adaptation processes: (i) infrastructure regulation, and (ii) governance refinement, itself being divided between (a) horizontal and (b) vertical reorganization.

Our governance agents are based on the Belief–Desire–Intention (BDI) architecture. A BDI agent follows plans which are triggered by goals to achieve, and/or by internal beliefs, depending on a specific context. Such a plan details how to achieve the goal through a set of actions to perform on the environment and/or sub-goals to fulfill.

We defined a common BDI reasoning template that each type of governance agent follows. Such a template defines the common plans the governance agents will follow, but the goals, the sub-goals and the actions are abstract and will be specialized for each type of agent (see Sections 7.2, 7.3, and 7.4).

To define such a reasoning process, we chose to base it on our scalability governance definition. In Section 1.3.1.4 (p. 31), we have stated that *"a system as scalable if, when scaling up one several dimensions, the requirements on its properties are satisfied, or if a bearable adaptation of the requirements is found (resulting from a balanced trade-off between several properties)"* (see Definition 3, p. 32), which can be achieved by setting up different regulation tactics, as stated in Section 1.3.3 (p. 36):

1. A *scale-up* tactic consists of increasing the performances without adding resources to the system –e.g. algorithmic optimization– or by a trade-off between the requirements of the system –i.e. enforcing one property while relaxing an other.
2. A *scale-out* tactic consists of adding more resources to increase the satisfaction of the requirements while increasing the costs –e.g. financial, complexity, overhead– of the system.
3. A *scale-down* tactic consists of reducing the number of resources when they are under-used to reduce the costs.

Thus, we use these principles for reasoning about the governance. Figure 7.1 describes the reasoning process for such a governance tactic agent.

1. **Governance Enactment.** For each requirement defined in the horizontal and vertical organization as an organizational goal, the agents have to ensure the M2M infrastructure fulfills it. That is, for each requirement the agents control the functioning of a corresponding set of resources of the infrastructure. If the system does not meet the requirements, the agent performs infrastructure reconfiguration to try to achieve the governance objectives, using *scale-up tactics*.

   If infrastructure regulation still fails to meet the governance objectives, *scale-out* and *scale-down* tactics might be performed. Then, the agents also need to analyze the strategy –i.e. the organization– and try to reorganize it. This reorganization can be vertical (Figure 7.1, 3 and b) or horizontal (Figure 7.1, 1 and a).

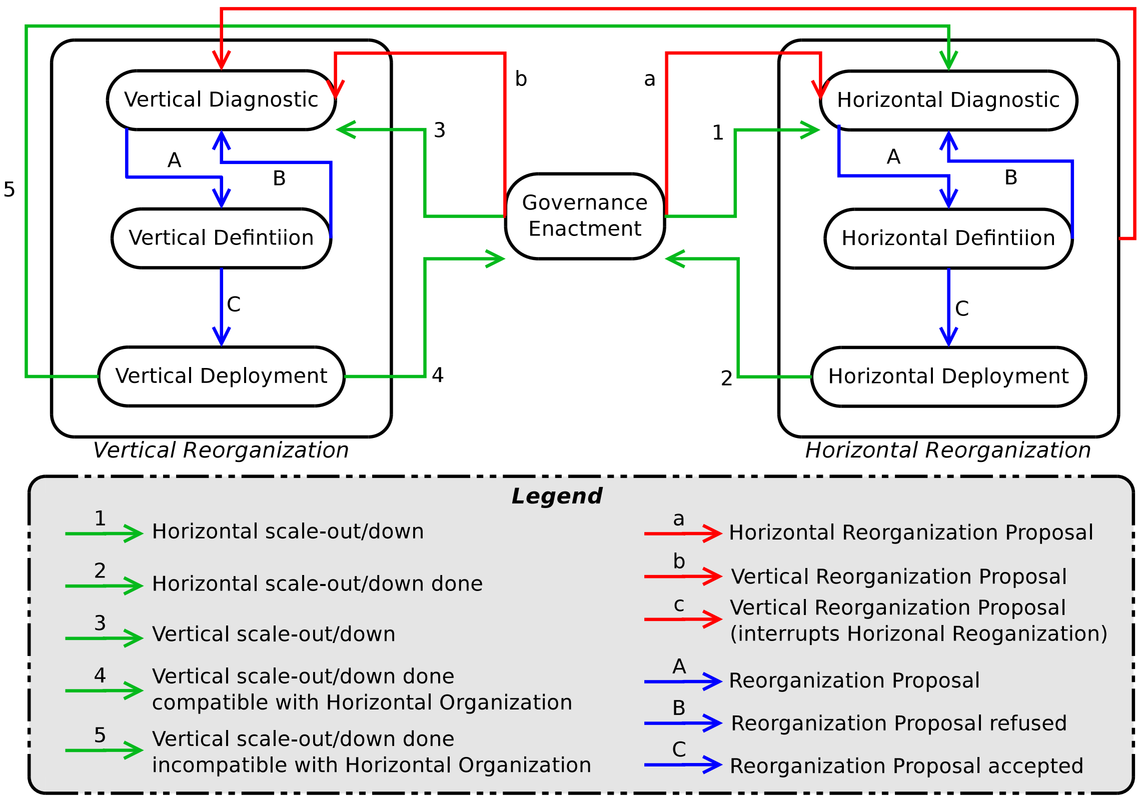

Vertical Diagnostic
Vertical Defintiion
Vertical Deployment
Vertical Reorganization
Governance Enactment
Horizontal Diagnostic
Horizontal Defintiion
Horizontal Deployment
Horizontal Reorganization
A
B
C
a
b
c
1
2
3
4
5
Legend
1 Horizontal scale-out/down
2 Horizontal scale-out/down done
3 Vertical scale-out/down
4 Vertical scale-out/down done compatible with Horizontal Organization
5 Vertical scale-out/down done incompatible with Horizontal Organization
a Horizontal Reorganization Proposal
b Vertical Reorganization Proposal
c Vertical Reorganization Proposal (interrupts Horizontal Reoganization)
A Reorganization Proposal
B Reorganization Proposal refused
C Reorganization Proposal accepted

Figure 7.1: Governance Agent's Reasoning Cycle.

2. **Horizontal Governance Reorganization.** This corresponds to an *horizontal governance refinement*. It is performed when new agents are needed to ensure the governance or when agents are not useful anymore. This is the case when an infrastructure scale-out or scale-down is required (Figure 7.1, 1 and a) or when a vertical SLA requires more agents to be deployed (Figure 7.1, 5).

   The agents determine the number of agents necessary for achieving a governance objective –i.e. a role, a mission or a goal. Based on this, new agents are created or terminated, and the tasks are dispatched among them. Then, the agents return to the regulation phase (Figure 7.1, 2).

   As the horizontal governance depends on the realization of the vertical governance, we prioritize the vertical governance. Thus, when a vertical reorganization is needed, the agents interrupts the horizontal one in order to take into account the new constraints.

3. **Vertical Governance Reorganization.** This corresponds to a *vertical governance refinement*. This phase can be triggered either (i) by a new proposal of a vertical organization (i.e. new SLA proposal), or (ii) because the agent cannot afford to find a way for the infrastructure to meet the requirements.

   The agents analyze the vertical requirement in order to determine whether the infrastructure can meet them or not. If not, the agents try to negotiate the terms of the SLA until a satisfiable compromise is found. Then, the new organization is deployed and the agents return to the regulation phase (Figure 7.1, 4).

If no agreement is found, this can be due to two causes: (i) no SLA refinement is possible (e.g. devices' load exceeded), then the vertical organization is dropped and the agents return to the regulation phase (Figure 7.1, 4), or (ii) an SLA agreement is found, but some agents are unable to handle it, then an horizontal reorganization should be performed first (Figure 7.1, 5).

7.1.1 Governance Enactment

Governance enactment is the first step of the governance agents' reasoning cycle. That is, the agents configure the M2M infrastructure, so that it respects the strategy requirements. If needed, such requirements are enforced by M2M infrastructure regulation. This consists of a modification of the infrastructure nominal functioning so that it can reach better performances.

The different types of agents will adopt different types of roles in the organizations (described in Chapter 6), therefore, the parts of the M2M system to monitor and control will differ from one to an other. Likewise, they will not manage the same parts of the SLA.

The common plan for the monitoring phase of an abstract objective consists of actually monitoring a set of M2M components. The agent observes infrastructure failures with respect to the strategy requirements, in order to detect regulation needs. Nevertheless, to avoid perpetual adaptations and ensure some stability, a tolerance threshold should be defined beneath which requirements failures will be tolerated by the agents.

When the number of failures overpass such a tolerance threshold, the agent trigger a regulation plan. Depending on the nature of the type of the failure and the concerned resource, the agent determines the regulation policy to use for handling the failures. Such a regulation can be implemented by either a *scale-up*, a *scale-out* or a *scale-down* tactic.

Scale-Up Tactics. Scaling up the infrastructure consists of a logical adaptation of the M2M infrastructure's behavior. As, the cost for such an adaptation is less important than infrastructure deployments, it is prioritized beside the scale-out tactic.

In fact, scaling up the M2M infrastructure can be done through simple regulation actions. Though this list could be extended, Table 7.1 describe the regulation tactics we took into account.

Such actions are implemented by the means of *policy artifacts*, and so we will detail this point in Chapter 8. Using an artifact facilitates the agent's tasks to set up, activate and deactivate policies, to monitor the efficacy of the infrastructure's regulation, and enables them fine tune the parameters of the policy. Thus, the agent does not need to monitor and analyze continuously each M2M resources, but it can observe the artifacts, check their performances and be alerted in case of failure.

Scale-Out Tactics. Scaling out the M2M infrastructure consists of adding more resources to satisfy the strategy requirements while handling an increased traffic. Such a deployment can be performed either by the agents directly –e.g. new platform using a pool of virtual servers– or may involve an human intervention –e.g. new M2M gateways or WSAN repeaters in the real world. In the latter case, the agents should be able to communicate such a recommendation to the human administrators.

Tactics	*Description*
`routing`	Define an alternative route or routing algorithm at the WSAN level
`threshold`	Define a threshold limit over which a component will reject all the requests
`enforce`	Enforce a policy parameter so that the M2M infrastructure complies more with the requirements
`relax`	Relax a policy parameter to save resources
`resilience`	Provide alternatives to a failing component of the M2M infrastructure

Table 7.1: Regulation Tactics used to Scale-Up M2M Infrastructure.

The scale-out tactic is performed using together the `deploy` and `load balance` actions among the duplicated components. Load balancing is performed using scale-up actions `enforce` and `relax` threshold values for each component.

Scale-Down Tactics. Scaling down tactics consists of reducing the governance overhead over the M2M infrastructure. We define scale-out strategies to reduce (i) the size of the M2M infrastructure and (ii) the governance overhead.

On the one hand, the agents evaluate the M2M components workload. When useless components are detected, the agents' scale-down tactic shuts them down to save resources. This is particularly important towards scale-out regulation in order to reduce the financial cost induced by the additional components.

On the other hand, a scale-down tactic is used to limit the overhead cost induced by the governance system itself. That is, regulation policies previously set up are observed by the agents to determine their utility. When a policy is considered as useless –i.e. it is rarely activated– the agents deactivate it.

Further more, the agents also evaluate their own utility in order to maintain an acceptable tactic layer size. To do so, the agents reorganize the tactic layer and redistribute the governance tasks.

From Regulation to Reorganization. As mentioned previously, each tactic has a different cost. On the one hand, while the *scale-up* tactic generates an overhead, cost due to the regulation computation, the *scale-out* tactic also generates a communication overhead, due to the balanced tasks, and a financial cost for the additional infrastructure deployment. On the other hand, the *scale-down* tactic allows to partly reduce such costs.

Therefore, it is necessary to prioritize the different tactics: *scale-out* should be performed after several tentatives of *scale-up* have failed, and *scale-down* when regulation have not been used for a while.

Further more, regulation might require a reorganization of the governance MAS. In fact, when scale-up regulation fails, this can be due to a misfit SLA that is not satisfiable. Hence, the agents should redefine the

SLA through a vertical reorganization, corresponding to a vertical governance reorganization. More over, scaling out the infrastructure requires more agents to insure the horizontal governance. Thus, an horizontal reorganization is triggered, corresponding to an horizontal governance refinement.

7.1.2 Vertical Governance Reorganization

The vertical reorganization process consists of an adaptation of the vertical strategy, that is, to manage the evolution of the vertical contracts. This process is initiated in two cases: (i) when a new vertical contract is introduced in the M2M infrastructure by the stakeholders, and (ii) when the infrastructure is incapable to meet an SLA and the governance regulation fails. In both cases, the governance agents have to pass through a reorganization process in order to evaluate and validate the corresponding SLA.

Such a reorganization addresses directly the specifications of the contract. So, it is an OS reorganization where the agents reason about the goals attributes. In this section, we describe the steps of such a reorganization, but we will detail the individual reasoning for each agents in Sections 7.2, 7.3 and 7.4. Figure 7.2 describe the common process for each governance agent.

1. To start the vertical reorganization process, a `Call For Participation` message is sent to the governance agents. Then, each agent evaluates its individual availability –i.e. its individual workload. The agents which participate in this process stop their governance regulation to focus on the reorganization. Let us focus on two particular cases:

   – If the reorganization addresses one of its commitments (i.e. an existing vertical organization), the agent directly participates in the reorganization.

   – If no agent of a kind replies to the `Call for Participation`, an horizontal reorganization is triggered in order to deploy a new agent, so that it can participate in the SLA validation.

2. When an agent of each type is found, they all evaluate the feasibility of the proposed SLA. To do so, each type of agent only evaluates parts of the SLA from its fields of expertise. It checks the feasibility of the proposed SLA with respect to the M2M infrastructure workload.

3. If an agent considers its part of the SLA to be impossible, it tries to find a bearable compromise. If it finds one, it proposes it to the other agents, which in turn evaluate it (Step 2).

   If no compromise is found, the SLA is reject, and the vertical reorganization is stopped.

4. When all the SLA is validated by the governance agents, they deploy it on the M2M infrastructure. This step consists of (i) creating an instance of a vertical organization corresponding to the validated SLA, (ii) committing to the different roles corresponding to the parts of the SLA the agents have taken the responsibility of, (iii) starting the different functional schemes, and (iv) configuring the M2M infrastructure with respect to the agreed SLA.

SLA Negotiation Issues. Such a vertical reorganization process allows our governance framework to handle new vertical contracts, but also their evolution. Nevertheless, the negotiation (Steps 2–3 loop) of

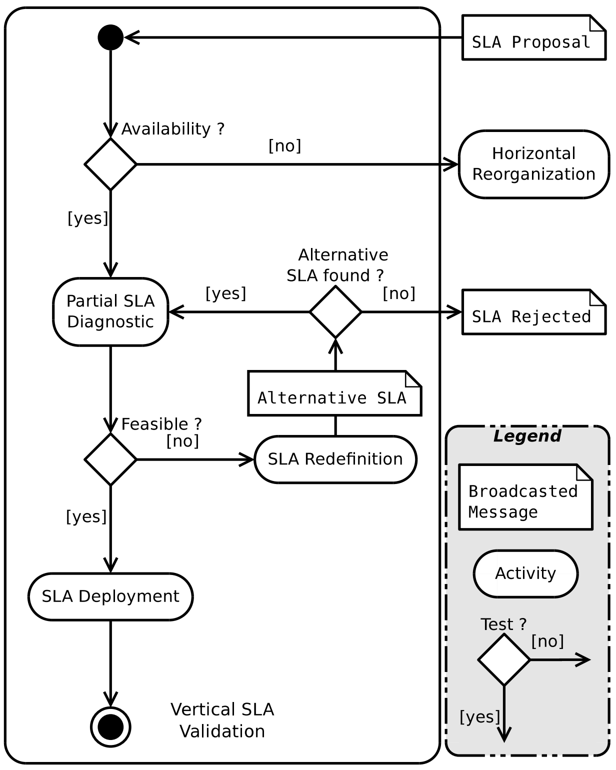


Figure 7.2: Vertical Reorganization Process.

the SLA implies other considerations, such as legal aspects, trust management or semantic descriptions, to mention these examples only.

As the terms of the SLA are originally set by a contractual agreement between the human stakeholders, the agents can hardly modify them without raising legal issues. A possible solution would be to introduce the human in the loop, so they validate the SLA modifications.

Another possibility would be for the different stakeholders to define a negotiation policy, so that the agents can use it to refine the SLA. In this case, the agents should then embody a trust mechanism to perform the automatic negotiation between the stakeholders.

Such an automatic negotiation would raise new possibilities for the governance of open systems, such as the M2M. For example, using a semantic description of the data and services provided by each M2M entity, it would be possible to dynamically find the M2M platforms and devices suiting to the application needs.

These aspects are still open issues in their respective communities and could be a research project on its own. Therefore, we do not address them in this thesis, but we chose to let the possibility for one to extend our model, by specifying the loop between Step 2 and Step 3.

7.1.3 Horizontal Governance Reorganization

The horizontal reorganization process consists of an adaptation of the horizontal strategy, that is, to manage the evolution of the whole governance MAS. As we define the horizontal OS in such a manner that agents have the maximum of liberty, we implement the horizontal refinement through an OE reorganization. Further more, this choice is motivated by an other consideration: the horizontal strategy is defined based on the ETSI TC M2M recommendations, therefore, we do not want to reconsider the ETSI TC M2M requirements.

Hence, the horizontal reorganization addresses the way the MAS will coordinate together to apply the horizontal OS. That is how, the number of agents will evolve and how they will share common goals to govern correctly the M2M infrastructure.

As such an horizontal reorganization does not address strategy adaptation but the tactic adaptation, we choose not to use an organization but a coalition. That is, we define an implicit organization –i.e. not specified in terms of roles and schemes– which the agents are aware of.

To do so, the agents of the same type will create a coalition. A coalition groups a set of agents $\mathcal{A}$, directed by a coordinator agent $\mathbf{a} \in \mathcal{A}$ that share a common goal $\mathbf{G}$. The coordinator agent is responsible for initiating the coalition, adding/removing agents in the coalition, coordinating the coalition and terminating the coalition. Such a coalition distributes the common goal's sub-goals among the agents which, in turn, can create a lower-level coalition.

Types of Coalitions. In Table 7.2, we describe how we use tactic coalitions to distribute the governance among several agents.

When an agent is overloaded, it determines the level of governance causes a problem. Then, it splits its horizontal commitments (**a.**), its vertical commitments (**c.**) or the governance of a specific SLA (**d.**) to delegate them to other agents in a dedicated coalition. Thus, it can focus more on specific requirements that need more attention.

In addition, in order to limit the agents' load, we distribute the governance of the different parts of the M2M infrastructure. Each external M2M entity –i.e. each application and each group of devices– is governed by a different agent, respectively an *AppAg* and a *DevAg*. And each part of the internal infrastructure –i.e. each core platform, each gateway– will also be governed by a different *InfAg* agent.

Thus, deployment (or departure) of a new part of the M2M infrastructure triggers an horizontal reorganization to deploy (respectively, terminate) the corresponding type of agents.

Scale-Out vs. Scale-Down. In order to ensure the agents to correctly govern the M2M infrastructure, we limit the governance load for each agent. *Scaling-out* the governance OE consists of deploying new agents within the actual coalition or initiating a new sub-level coalition. *Scaling-down* the governance OE consists of terminating agents in the coalition or a coalition itself.

Hence, we define an heuristic that allow the agents to decide whether to scale-out or scale-down the governance OE, based on own workload.

Coalition Goal	*Description*	*Individual Goal*
a. Govern the horizontal requirements	Groups agents responsible for the same horizontal governance	Govern a sub-set of the horizontal requirements (roles)
b. Govern several M2M entities	Groups agents sharing the same horizontal objectives	Govern a single entity
c. Govern vertical contracts	Groups agents representing the same M2M entity	Govern a single vertical commitment
d. Govern a vertical SLA	Groups agents responsible for the same vertical commitment	Govern a sub-set of the SLA

Table 7.2: Granularity Levels of Governance Coalitions.

Agents' Coordination within a Coalition. While duplicating governance agents, several agents share the same goals, and so, they need to coordinate. Hence, they can distribute among themselves the commitment to different goals and the different vertical roles. That is, each agent will focus on a particular task, and ask the others to achieve its other goals.

To do so, we use a greedy algorithm to determine the less loaded agent that should commit to the goal: when an agent share a role with others, it delays its commitment during a time T_{delay}, inversely proportional to its workload:

$$T_{delay} := TC - TC \times \frac{ML - EW}{ML} \tag{7.1}$$

Where, TC is the time to commit, ML is the agent's individual maximum limit, EW is the expected agent's workload if it commits to the goal. Hence, T_{delay} will tend to TC as much as EW is close to ML, and so the agent will wait longer before committing. If $EW > ML$, the agent will never commit before the deadline. Finally, the agent which greater $ML - EW$ difference –i.e. the most free agent– will commit first.

When no agent commit to the mission or the role, that means all the agents are overloaded, and so, a new agent should be created. To do so, we consider the first agent created of its type as the coordinator agent for the agent duplication.

7.1.4 Agent Specialization

After describing the base behavior of the governance agents, we now explain how we define the different specialized types of agents. As mentioned previously, the agents are representative for the interests of the different stakeholders involved in the end-to-end M2M infrastructure: (i) the M2M application provider (*AppAg*), (ii) the M2M devices provider (*DevAg*), and (iii) the M2M Telco infrastructure provider (*InfAg*) (see Figure 7.3).

That is, each type of specialized governance agent will focus on both the horizontal and the vertical considerations for a specific part of the infrastructure. Hence, we define how the agents bind the horizontal requirements to the vertical requirements.

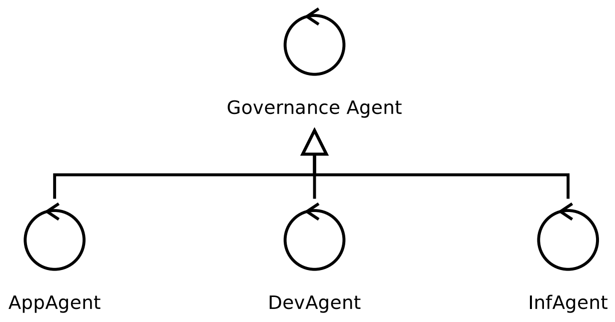


Figure 7.3: Each specialized type of agent is driven by a different focus on the M2M infrastructure.

On the one hand, the application and the device agent consider the strategy objective from the external stakeholders point of view. That is, their objective is to govern a specific stakeholder's interests. And so, they have an horizontal view of a specific stakeholder's vertical implications. Therefore, they participate only in the horizontal roles that are dependent to their stakeholder's point of view.

On the other hand, the infrastructure agent consider the strategy objective from the Telco point of view. That is, its objective is to serve as much as possible the other stakeholders. Hence, it is horizontally implicated in all the vertical contracts, from the infrastructure point of view.

Further more, each type of agent will support these horizontal and vertical requirements regarding the stakeholders' specific interests. That is, requirements that are specific to each type of stakeholder.

Based on these principles, we can bind the horizontal and vertical requirements at the governance tactic level, as we did for the strategy in Section 6.4. The meaning of such a binding is each the horizontal requirement is satisfied by all the instances of the corresponding vertical requirement.

We report such binding in Table 7.3. For clarity purpose, we aggregate the three *sensor*, *actuator* and *repeater* requirements as a single *device* requirement in this table, as their difference is not relevant from the horizontal stand point.

Such a table allows us to define the individual responsibilities of the different types of governance agents. We assign each consistent set of horizontal/vertical requirements to a different type of agent. Let us notice that some horizontal requirements are dependent to several vertical requirements, and possibly implicates different types of agents (e.g. *Compensation and Brokerage*). In fact, some of these requirements are cross-wide along the whole M2M infrastructure, and so, should be propagated along several vertical levels.

For each type of agent, we describe, in the next sections, their horizontal and vertical roles, their regulation tactics, and their horizontal and vertical reorganization reasoning.

7.2 Application Agents

Application agents (*AppAg*) ensure the requirements fulfilment from the third-party application point of view. Thus, this type of agent regulates the commands and requests sent by an application to the devices and check the notifications it receives.

Vertical / Horizontal	*application*	*platform*	*gateway*	*device*
Application Enablement	*AppAg*			
Reachability, Addressing and Repository				*DevAg*
Communication Selection				*DevAg*
Remote Entity Management				*DevAg*
Transaction Management				*DevAg*
Generic Communication		*InfAg*	*InfAg*	*DevAg*
Telco Operator Exposure		*InfAg*		
Interworking Proxy		*InfAg*		
History and Data Retention		*InfAg*		
Security	*AppAg*	*InfAg*	*InfAg*	
Compensation and Brokerage	*AppAg*	*InfAg*	*InfAg*	*DevAg*

Table 7.3: Horizontal and Vertical requirements binding and the corresponding affectation to the agents

Application-Driven Governance. One *AppAg* agent manages the governance for, at most, one client application, that is an application that subscribe to one or several groups of devices, using an unique stakeholder identifier.

As identified in Table 7.3, the *AppAg* agent participates in the realization of the following requirements: *Application Enablement*, *Security* (partially) and *Compensation Brokerage*; and the following vertical requirement: *Application*.

Stakeholder's Objectives. The main objective of an M2M client application is (i) to *maximize the utility of M2M devices* it subscribes to, whilst (ii) *minimizing the cost* induced by the subscriptions and the contract violation. Thus, the *AppAg* agents' activities are defined taking into account these consideration.

In this section, we describe how we use the *AppAg* agents to bind the horizontal and vertical requirements (Section 7.2.1) to regulate the applications to make it meet the requirements (Section 7.2.2) and to refine the strategy to take into account the application needs (Section 7.2.3).

7.2.1 Horizontal / Vertical Requirements Binding

We start by analysing the horizontal sub-requirements –i.e. organizational goal level– to determine how the *AppAg* agents can fulfill them in each vertical involvement.

The *Application Enablement* requirement (*rAE*) consists of ensuring the client applications an access to the different M2M services. This requirement is validated if every individual application has an effective access to the M2M services.

The *gAEExpose* and *gAERouting* requirements are satisfied by the effective realization of each vertical subscription, while the *gAERegister* is satisfied by allowing client application to register to the different

M2M services and the *gAERecord* requirement is validated by pertaining histories for each vertical subscription.

Such requirements are fulfilled vertically by the *deliverToApp* requirement.

These requirements do not need much intelligence to be fulfilled, therefore the *AppAg* agent only have to ensure that the corresponding services are activated on the core platform. If these services are not activated or crash, the agent will simply tell the *InfAg* agent to restore these core platform services (see Section 7.4).

The *Security* requirement (*rSec*) consists of ensuring the security of the M2M infrastructure. Such a requirement is partially addressed by the application and authorization management (*gAEAuth*).

Such requirement is fulfilled by the validation of each vertical *authCmdSender* and *checkAppRights* requirements.

The *Compensation and Brokerage* requirement (*rCB*) consists enabling the M2M infrastructure to dynamically handle the introduction of new vertical contracts and generating compensation fees based on the SLA realization and violation.

While the latter is handled by the vertical *gComputePenalties* and *gBillCompensation* goals, the former is performed by the *AppAg* vertical reorganization tactic (see Section 7.2.3).

7.2.2 Application Regulation

We now describe the plans used by the *AppAg* agent to govern the applications. Such an agent controls both the client application satisfaction from the M2M services and its compliance to the vertical SLAs.

M2M Devices Subscription, Compensation and Penalties. At first, the *AppAg* monitors the application's access to the M2M devices it has subscribed to, and check its validity towards the SLA attributes.

In a *Data Collection* scheme, such an access is governed by the *notifyApplication* organizational goal, with the `[max_latency]` attribute. And, in a *Command Sending* scheme, it is specified by the *sendCommand* organizational goal, with the `[command_rate]` attribute.

In the case of SLA violation, the *AppAg* will charge the penalties to the corresponding stakeholder, as defined in the *Compensation and Penalties* scheme (*gComputePenalties*). That is, it will charge the application stakeholder if it sends too many commands, as it could drop down the M2M devices' lifetime, or the platform, and/or charge the device stakeholders if the devices data exceed the tolerated latency or loss.

The compensation itself, is computed when the vertical contract comes to an end. To do so, the agents use the compensation functions provided with the contract and the observed violations.

Application Rights Enforcement. In order to fulfill the *authCmdSender* and *checkAppRights* requirements, the *AppAg* agent must control the client application access rights. Such a control is abstracted by the artifact layer, as described in Chapter 8, so that the agent can focus on the governance reasoning process.

Table 7.4 describes the regulation plan implemented by the *AppAg* agent. Such a plan, and the followings in the remainder of this thesis, should be read as follows: *g* denotes the root goal of the plan, `[PARAM]` denotes a QoS parameter, *.action(...)* denotes an action executed by the agent, *!goal* denotes a sub-goal to satisfy, *?match(V)* is an evaluation goal that resolve an association between one or several variables V.

Goal:	*check_app_rights* `[APP_ID,DEV_ID,RQ_MAX_CMD]`

Enactment:	.configure_app_rights(APP_ID,DEV_ID)
	.monitor_app_access(APP_ID)

Failure: Regulation:	Cmd_Freq > RQ_MAX_CMD + T .set_up_threshold_interception(RQ_MAX_CMD)
Observed: Relax:	intercepted_abused < T .relax_threshold_interception
Observed: Regulation:	intercepted_abused = 0 .stop_threshold_interception

Table 7.4: Regulation Plan to control the Applications Rights.

In this case, the plan enables the agent to enforce the application rights compliance, as follows.

In order to prevent the application from exceed too much their rights, the *AppAg* agent can set up a `threshold` control to intercept the application exceeding commands. Though, an excess tolerance value T is determined, to allow the application to deal with exceptional needs and urgent commands, over which the agent activate the `threshold` control.

Such a threshold value is fixed to the exact SLA value to enforce the application compliance with the SLA. Nevertheless, if the client application regulates its behavior during a while, the *AppAg* agent can progressively relax the constraint to the tolerated excess T. Finally, the agent would stop the active control when it becomes unnecessary.

After the definition of the *AppAg* agent's nominal regulation governance tactics, we focus on its behavior regarding the strategy refinement (reorganization), in the next section.

7.2.3 Reorganization

The *AppAg* agent's refinement tactic can be considered as a reorganization in both the vertical and the horizontal dimensions.

The horizontal reorganization consists of (i) adding/removing *AppAg* agents corresponding to the number of registered applications, as described in Section 7.1.3, and (ii) initiating/terminating the vertical organizations corresponding to the activated vertical contracts.

Hence, the *AppAg* agents' strategy refinement is mostly driven by the vertical concerns.

As defined in Section 7.1.2, the vertical reorganization is performed to tackle two situations: (i) the introduction of a new vertical contract, and (ii) the inadequacy of the M2M infrastructure's behavior regarding the SLA.

The latter case, from the *AppAg* governance agent point of view, would be addressed if the client application tried to send too many unauthorized request. This is handled using a threshold regulation policy, as described in the previous section. Except for the definition of a new SLA contract, there is no need for the *AppAg* to reorganize such a definition.

Thus, the *AppAg* agent focuses on the introduction of new vertical contracts.

SLA Proposal. When a client application requests an access to M2M devices, it submits an SLA proposal. It is handled by the *AppAg* agent responsible for the corresponding stakeholder. It starts a *recruitment* scheme, so that the governance agents validate (or not) and deploy the corresponding vertical management on the infrastructure.

SLA Validation. As the SLA proposal is defined by the client application stakeholder, the *AppAg* agent does not need to evaluate its feasibility, from the application point of view.

Nevertheless, it evaluates the load that would generate such an additional governance. If it expects it to be too important, the agent would delegate it to another agent (see Section 7.1.3).

SLA Deployment. As the *AppAg* agent is the initiator of the SLA validation process, it is the one that create and instantiate the vertical organization and, when all the agents have joined it, starts the schemes.

For the SLA deployment itself, it consists of configuring the artifacts, that abstract the core platform components interfacing client application with the M2M infrastructure, to be notified of the SLA violations.

7.3 Device Agents

Device agents (*DevAg*) control the usage of the M2M devices –i.e. sensor and actuators performing applicative tasks, and repeaters.

Device-Driven Governance. One *DevAg* agent manages the governance for, at most, one consistent group of M2M devices, that is devices providing the same service, in the same physical area and belonging to the same stakeholder.

As identified in Table 7.3, the *DevAg* agents participate in the realization of following horizontal requirements: *Generic Communication*, *Reachability, Addressing and Repository*, *Communication Selection*, *Remote Entity Management*, *Transaction Management* and *Compensation Brokerage*; and the following vertical requirements: *Sensor*, *Actuator* and*Repeater*.

Stakeholder's Objectives. The main objective of an M2M device provider is (i) to *maximize the number of client applications* it provides, whilst (ii) *maximizing the M2M devices lifetime*. Thus, the *DevAg* agents' activities are defined taking into account these consideration.

In this section, we describe how we use the *DevAg* agents to bind the horizontal and vertical requirements (Section 7.3.1), to regulate the M2M devices to make it meet the requirements (Section 7.3.2), and to refine the strategy to take into account the devices' constraints (Section 7.3.3).

7.3.1 Horizontal / Vertical Requirements Binding

We start by analysing the horizontal sub-requirements –i.e. organizational goal level– to determine how the *DevAg* agents can fulfill them in each vertical involvement.

The Generic Communication requirement (*rGC*) consists of ensuring the communication to the M2M gateway and the M2M devices. And so, the *DevAg* agents partially fulfill the requirements in collaboration with the *InfAg* agents committed to the *gateway* roles.

The *gGCDelivery* is satisfied by the fulfilment of each vertical *route_to_device* goals in the *Data Collection* scheme, and the *gGCTraffic* goal by the fulfilment of the SLA attributes `[message_frequency]`, `[max_loss]`, `[max_latency]` and `[weight]` in the *Data Collection* and *Command to Device* schemes, as well as the goal *deliverResponse* in the latter scheme.

The Reachability, Addressing and Repository requirement (*rRAR*) consists of managing the M2M device groups, routes to these devices, and subscription. This requirement is fulfilled by the governance of each vertical group tactic.

In our governance tactic, we decide to split each M2M device group vertically and to assign it to a different. Hence, each vertical contract addresses one and only one device group. The agents are responsible for governing all the vertical contracts for all the M2M groups.

The Communication Selection requirement (*rCS*) consists of selecting the most appropriate route in WSAN in a set of alternative routes.

Hence, the *gCSAlterRoute* goal is fulfilled vertically by the *route_to_device* and *route_to_gateway* goals.

The Remote Entity Management requirement (*rREM*) consists of ensuring the M2M management functions and protocol. Thus, it can be considered as *Command to Device* vertical scheme, for which the client application is the M2M device stakeholder (or manager) and the commands are either a firmware upgrade, a management function or a management protocol.

The Compensation Brokerage requirement consists of ensuring the financial compensation corresponding to the vertical contracts. So, the *DevAg* participates in this role, representing the devices involved in the contract.

Hence the *gCBToken* goal depends on the *gDeployGov* vertical goal, the *gCBValid* goal by the *gRecruitGov* goal, and the *gCBBill* and *gCBRedeem* goals by the *gCompensate* goal.

7.3.2 Device Regulation

We now describe the plans used by the *DevAg* agent to govern the devices. Such an agent tries to make the devices meet the vertical strategy requirements, whilst ensuring the devices' properties –i.e. the lifetime requirement.

Goal:	*sense_environment* `[RQ_SENSING_RATE]`
Enactment:	.configure_devices(SENSE,RQ_SENSING_RATE) .monitor(Sensing_Rate)
Failure:	Sensing_Rate > RQ_SENSING_RATE + ET *[ET: Enforcement Tolerance]*
Regulation:	.enforce_sensing_rate(EF) *[EF: Enforcement Factor,* $0.9 < EF < 1$*]*
Failure:	Sensing_Rate > RQ_SENSING_RATE & Enforcement_Trials > MAX_TRIALS
Refinement:	!vreorg(replace(RQ_SENSING_RATE,Sensing_Rate))

Table 7.5: Regulation Plan to control the sensors behavior.

As described previously, the *DevAg* agent's horizontal goals can be expressed in terms of vertical objectives fulfillment. Thus, in the remainder of this section, we focus on the agent's vertical goals.

Environment Sensing. For each vertical contract specifying a *Data Collection* scheme, the *DevAg* agent deploy and ensure the *sense_environment* goal. In Table 7.5, we describe the regulation plan used by the agent to control and ensure this requirement.

First, the agent configure the devices with respect to the vertical SLA. Then, it monitors the QoS value (Sensing_Rate) to verify its compliance with the defined requirement. If a failure occurs and overpass a tolerance threshold, the agent tries to enforce the SLA satisfaction, by reconfiguring the M2M devices with a lower sensing rate value.

If the requirement failures persists, after a several enforcement trials, the agent change the tactic. Instead of trying to make the M2M devices meet the SLA, it will try to refine the requirement to the actual infrastructure's performances. And so, it proposes a *vertical reorganization*, to replace the SLA with the measured QoS value.

Route Selection. The *DevAg* agent is responsible for selecting the most accurate route to be used in the WSAN for the message transmission to fulfill the horizontal *gCSAlterRoutes* and the vertical *route_to_devices* and *route_to_gw* goals. Hence, the *DevAg* agent uses a `routing` tactic, to ensure the `[RQ_LATENCY]` and the `[RQ_LOSS]` QoS, together with an `enforcement` tactic to ensure the `[RQ_MSG_FREQ]` QoS.

Table 7.6 gives the *DevAg* agent's plan for ensuring the *route_to_devices* objective. We do not detail the plan for the *route_to_gw* goal, as it is very similar to this one. Neither we do for the `[RQ_LOSS]` enforcement and reorganization, as it follows the same procedure as for `[RQ_LATENCY]`.

First, the *DevAg* agent selects the route that provides the maximal viability with a theoretical acceptable latency (SR) among the candidate routes (CR). Then, it configures the M2M devices with the selected route (command "$ROUTE$"). That is, the one that use the M2M devices with the highest lifetime. Hence, it

Goal:	*route_to_devices [RQ_MSG_FREQ,RQ_LATENCY,RQ_LOSS]*

Enactment:	.configure_devices(NOTIF,RQ_MSG_FREQ)
	?select_route_max_viability(CR,SR)
	.configure_devices("ROUTE",SR)
	.monitor(Msg_Freq,Latency,Loss)

Failure: Regulation:	Msg_Freq > RQ_MSG_FREQ + ET enforce_message_frequency(EF)
Failure: Refinement:	Msg_Freq > RQ_MSG_FREQ + ET & Enforcement_Trials > MAX_TRIALS !vreorg(replace(RQ_MSG_FREQ,Msg_Freq))
Failure: Regulation:	Latency > RQ_LATENCY + ET ?select_route_lower_viability(CR,SR) .configure_devices("ROUTE",SR)
Failure: Refinement:	Latency > RQ_LATENCY + ET & no_satisfying_route_in_Vorg !hreorg(find_other_routes(NR,RAgs)) !vreorg(hire_repeater_candidates(RAgs)) .configure_devices("ROUTE",NR)
Failure: Refinement:	Latency > RQ_LATENCY + ET & no_satisfying_route !vreorg(replace(RQ_Latency,Latency))

Table 7.6: Regulation Plan to manage the WSAN routing.

privileges the *lifetime* property over the *latency* property.

If the WSAN fails to meet the *[RQ_LATENCY]* requirement, then the *DevAg* agent tries to regulate it. To do so, it select another route that have a better theoretical latency but a worse viability.

When all the routes have failed to meet the requirements, the agents triggers an horizontal reorganization in order to hire new repeater devices, that could increase the WSAN performances. Such an horizontal reorganization also requires a vertical reorganization to include the new repeaters' representative agent into the vertical organization.

Finally, if no satisfying route is found to meet the SLA, a vertical reorganization is triggered. Hence, the *DevAg* proposes to replace the *[RQ_LATENCY]* value with the more realistic *[Latency]* value.

These tactics enable the *DevAg* agents to align the M2M devices usage with the horizontal and vertical strategy objectives.

As mentioned previously, when the agents fail to align the strategy requirements with the M2M devices constraints, they try to refine the strategy to make it comply with the resources' constraints. We discuss such related aspects in the next section.

7.3.3 Reorganization

The *DevAg* agent's refinement tactic can be considered as a reorganization in both the vertical and the horizontal dimensions. We do not detail the horizontal reorganization here, as it follows the description given in 7.1.3.

The vertical reorganization enables to take into account the devices' constraints at the strategy level. In fact, the *DevAg* agent participate in this process in order to "defend" the devices' interests. That is, the M2M devices should participate in the maximum of vertical contracts, increasing their benefit, until it does not drain down their energy lifetime too much.

To do so, the *DevAg* agents evaluate the vertical SLAs using the devices' energy consumption model and compare it with the devices' lifetime objective. Such an energy consumption model and lifetime objective is provided to the agent by the lower level abstraction layer (artifacts) that will be described in Chapter 8.

Such an evaluate occurs when (i) a new vertical SLA is submitted, and (ii) the M2M devices continuously fail to meet the requirements, even after several regulation trials.

New SLA. When a new vertical proposal is submitted, the *DevAg* agent evaluate the SLA's QoS attributes to decide whether to validate or reject them. To avoid conflicts, vertical commitment proposals are treated sequentially. For each QoS attribute, we can identify three cases:

1. The QoS value is less restrictive than the actual device's activity –e.g. due to other vertical commitments– then the attribute is validated.

2. The QoS value is more restrictive than the actual device's activity but comply with the devices' lifetime objective, then the attribute is validated, but it will be refined in priority when vertical reorganization is needed.

3. The QoS value does not comply with the devices' lifetime objective, then the attribute is rejected. The agent can, eventually, propose to negotiate such a QoS value to a less restrictive one.

Routes Refinement. If all the attributes are validated, the vertical SLA is validated too. Before adopting the corresponding role in the vertical organization and deploying the SLA, the *DevAg* agent may need to recruit M2M repeaters to enable the end-to-end communication.

That is, the agent selects one or several possible routes, that allow the M2M devices to communicate with the M2M gateway, and forwards the SLA proposal to the corresponding *DevAg* agents. These agents validate (or reject) in turn the proposal, as described previously, and join the vertical organization.

This route also determine the potential gateways to be used. Therefore, the *DevAg* agents participating as a *repeater* hire the corresponding *InfAg* to participate in the corresponding vertical organization as a *gateway*.

Such a recruitment can also be triggered by the routing regulation described previously.

SLA Update. When the governance agents are unable to make the M2M devices fulfill a vertical sla, they try to redefine the vertical contract.

The *DevAg* agent selects attributes that are unfulfilled and propose to the other agents of the corresponding organization to renegotiate them. For each attribute, it proposes a less restrictive value. Such a value can be determined based on another vertical SLA the M2M devices are involved, and which are correctly fulfilled. If no other acceptable value is known by the agent, it uses an heuristic to arbitrarily define it.

7.4 Infrastructure Agents

The Telco infrastructure agents (*InfAg*) are responsible for the Telco infrastructure functioning. This comprises all the infrastructure elements managed by the Telco operator, i.e. the core platform, but also the gateways. They contribute to the governance by evaluating and managing the traffic and the load generated on the infrastructure.

In this context, we assume the Telco stakeholder use a set of several servers and several gateways in order to guarantee the resilience of the M2M transmissions.

Infrastructure-Driven Governance. One *InfAg* agent manages the governance for, at most, one entity of the Telco infrastructure, that is, a core platform server or an M2M gateway.

As identified in Table 7.3, the *InfAg* agents participate in the realization of the following horizontal requirements: *Generic Communication*, *Telco Exposure*, *History and Data Retention*, *Interworking Proxy*, *Security* and *Compensation and Brokerage*; and the following vertical requirements: *platform* and *gateway*, depending on the M2M entity the *InfAg* governs.

Stakeholder's Objectives. The main objective of a Telco operator is (i) to *ensure the end-to-end transmission* of M2M messages, whilst (ii) *minimizing the infrastructure cost*. Thus, the *InfAg* agents' activities are defined taking into account these consideration.

In this section, we describe how we use the *InfAg* agents to bind the horizontal and vertical requirements (Section 7.4.1), to regulate the infrastructure to make it meet the requirements (Section 7.4.2), and to refine the strategy to take into account the infrastructure constraints (Section 7.4.3).

7.4.1 Horizontal / Vertical Requirements Binding

We start by analyzing the horizontal sub-requirements –i.e. organizational goals level– to determine how the *InfAg* fulfills its vertical commitments.

The *Generic Communication* requirement (*rGC*) consists of ensuring the communication to the M2M gateway and the M2M devices. And so, the *InfAg* agents, committed to the *gateway* roles, partially fulfill the requirements in collaboration with the *DevAg* agents committed to the repeater roles.

Whatever the vertical contract is, the *InfAg* agents ensure the session establishment with the M2M gateways (*gGCSession*), the communication encryption (*gGCEncrypt*), the message delivery (*gGCDelivery*) and failures report (*gGCReport*).

The *History, Data, Retention* requirement (*rHDR*) consists of archiving relevant information pertaining to messages exchanged, based on the other M2M capabilities' needs.

The *InfAg* agent manages the selection of information to store (*gHDRSelect*) and ensures the core platform correctly archives the corresponding data (*gHDRStore*), for all the vertical contract it is involved to govern the *platform* requirement.

The *Telco Operator Exposure* requirement (*rTOE*) consists of enabling several M2M core platforms to use the capabilities of each.

Across, the vertical deploy, the *InfAg* agents, committed to the *platform* requirement governance, manage the usage of all the deployed core platforms together, and so, can decide to delegate some functions to another server.

The *Interworking Proxy* requirement (*rIP*) consists of enabling non ETSI TC M2M compliant M2M entities to interact with the M2M core platform.

This requirement is ensured by the artifact abstraction layer, which allow the *InfAg* agents to dynamically manage the deployment of components and interfaces, redirect the calls to different the services (see Chapter 8).

The *Security* requirement (*Sec*) consists of supporting M2M service bootstrap (*gSecBootstrap*) and the key management (*gSecKeyMngt*) for mutual authentication (*gSecAuth*), and integrity validation (*gSecIntegrity*).

The *InfAg* agents govern the M2M service bootstrap through the artifact abstraction layer, as mentioned in the previous paragraph. The three last security concerned are managed by the *InfAg* agents by ensuring the correct functioning of security components on the core platform, they are responsible of.

The Compensation Brokerage requirement consists of ensuring the financial compensation corresponding to the vertical contracts. So, the *InfAg* participates in this role, representing the Telco infrastructure involved in the contract.

Hence, the *gCBToken* goal depends on the *gDeployGov* vertical goal, the *gCBValid* goal by the *gRecruitGov* goal, and the *gCBBill* and *gCBRedeem* goals by the *gCompensate* goal.

We can observe that, contrary to the *AppAg* and *DevAg* agents, the *InfAg* agents are driven by the horizontal concerns more than the vertical ones. That is, they fulfill the vertical *Platform* and *Gateway* requirements by ensuring the horizontal ones.

In fact all these requirements must be satisfied so that the *InfAg* can fulfill the vertical *infrastructure* sub requirements –*deliverToApp* (in the *Data Collection* scheme) and *deliverToWSAN* (in the *Command Sending* scheme)– and the *gateway* sub-requirements –*notifyPlatform* (in the *Data Collection* scheme) and *deliverToWSAN* (in the *Command Sending* scheme).

In the next section, we define the *InfAg* agents' tactic to ensure the correct functioning of the M2M core platforms and gateways, with respect to the horizontal requirements.

7.4.2 Infrastructure Regulation

We now describe the plans used by the *InfAg* to regulate the Telco infrastructures in order to ensure all the vertical instances to work correctly.

As described previously, the *InfAg* agents' vertical goals can be expressed in terms of horizontal objectives. Hence, in the remainder of this section, we focus on the agents' horizontal goals.

Requirement Resilience. In order to ensure that the core platform P it is governing comply with the horizontal requirements RQs, the *InfAg* agent starts by monitoring, through artifacts, the platform's functionalities Fs that implement the corresponding requirements.

Table 7.8 describes the regulation plan set up by the agent to control and ensure the resilience of an horizontal requirement RQ. That is, when it detects a functionality F that is unable to run correctly on the governed core platform, the agent tries to distribute the platform's tasks, as follows:

1. it searches for an alternative platform P' that is able to treat some of the P platform's tasks,
2. when, the agent finds another core platform P', that is able to ensure the corresponding functionality, it configures the corresponding artifacts to distribute the functionality F between the two platforms,
3. then, it activates the load balancing regulation policy between the servers.

Such a plan allows the *InfAg* agents to improve the resilience of the horizontal requirements' compliance. If no alternative platform is found, the *InfAg* triggers an horizontal reorganization in order to deploy a new core platform server, which can be shutdown latter (see Section 7.4.3). Hence, the agents can also dynamically adapt the platform resources depending on the M2M demand.

Nevertheless, if the reorganization process fails to deploy a new platform, the agent relaxes the corresponding requirement RQ –i.e. increase the requirement's failure tolerance– and triggers the payment of the penalties in each vertical contract impacted.

Gateway Resilience. Gateway resilience involves the same reasoning as the platform resilience. Nevertheless, an M2M gateway is physically deployed server, that cannot be deployed or shut down at will. So, the agents can only take into account deployed servers to distribute the gateway tasks.

Further more, M2M gateways are constrained by the area of WSAN the cover. Thus, not any server can be chosen as an alternative. In fact, to be an alternative candidate, the M2M gateway must provide a route to the M2M devices concerned by the tasks distribution.

As a consequent, such a regulation must be supported by a vertical reorganization, as described in the next section.

7.4.3 Reorganization

Vertical SLA Validation. A vertical SLA is validated by, at least, one *InfAg* agent for *platform* requirement and one *InfAg* agent for the *gateway* requirement. Each agent evaluate the possible load corresponding to the SLA and evaluate the capacity of the Telco infrastructure entity they govern.

If all the agents responsible for the M2M core platforms (or the M2M gateways) reject the SLA, the coordinator agent evaluates the possibility of deploying a new Telco infrastructure.

Goal:	*ensure_requirement(RQ,P)*
Enactment:	?resolve_platform_functionalities(RQ,Fs) .monitor_functionalities(Fs)

Failure:
Regulation:
	failing_functionality(F)
	?find_alt_platform(F,P')
	.distribute_functionality(F,P,P')
	.activate_load_balancing(P,P')

Failure: Regulation:	failing_functionality(F) & no_alt_platform_found !hreorg(deploy_new_platform) !failing_functionality(F)
Failure: Regulation:	failing_functionality(F) & failed(deploy_new_platform) ?functionality_requirement(F,RQ) .relax_requirement(RQ) !update_penalties.

Table 7.7: Regulation Plan to distribute functionalities over several core platforms.

Platform Deployment Utility vs. Cost. To help such a decision, we define a *Deployment Profit Heuristic* $dph(d)$ that defines whether the deployment would be *profitable* or *not_profitable*, based on (i) an history of the penalties paid for vertical failures, (ii) the potential loss of the previously refused contracts, (iii) the additional capacity that would be provided by the deployment, and (iv) the cost of such a deployment.

If the deployment seems to be profitable, the agent triggers it and initiate a new *InfAg* agent to manage the resource's governance and tells it to assume the governance responsibility in the new vertical contract. Finally, it enforces the horizontal requirements that might have been relaxed previously.

Scale-Down Decision. In order to save resources, the agents should also scale-down the Telco infrastructure. Such a decision is driven by the similar $dph(d)$ deployment heuristic, where the deployment d have a negative additional capacity and a negative cost.

The *InfAg* perform such an evaluation in two cases: (i) when a vertical contract come to an end, reducing so the future infrastructure workload, or (ii) when the governance artifact notifies the agents that the governed Telco entity is being underused.

7.5 Transition

```
* Base architecture for Governance Tactic Agent
 - Monitoring
```

Goal:	*respond_V_commitment(V_CONTRACT)* [my_governed_infrastructure(I)]
Context:	?capacity_evaluation(expected_load(V_CONTRACT), I, affordable)
Decision:	.accept_V_commitment(V_CONTRACT, I)
Context:	?capacity_evaluation(expected_load(V_CONTRACT), I, not_affordable)
Decision:	.refuse_V_commitment(V_CONTRACT, I)
	[Coordinator Agent]
Context:	refused_contract(V_CONTRACT), all) & ?dph(V_CONTRACT, profitable)
Decision:	!hreorg(deploy_new_inf) ?new_inf_ag(NIA) .tell(NIA, accept_V_commitment(V_CONTRACT)) !enforce_requirements
Context:	?capacity_evaluation(expected_load(V_CONTRACT), not_affordable) & ?dph(V_CONTRACT, not_profitable)
Decision:	.refuse(V_CONTRACT)

Table 7.8: Decision Plan for SLA Validation.

```
 - Reasonning => Adaptation
  - M2M Infrastructure Regulation => Governance Policies + Refinement
  - Scalability => NFR trade-offs
  - Misfit Strategy => Reorganization
   - OE-level: card of agents, goal distribution, and coordination
   - OS-level: SLA redefinition

* Synthesis of required resources :
 - monitoring/control of the infrastructure
 - governance policies control

* Abstraction needed :
 - governance policies
 - scalability concerns (NFR/Dimensions representation)
 - infrastructure representation
=> Environment of the Governance MAS => artifact-based abstraction
```

Chapter 8

Artifact-based Governance Policy

Contents

In this chapter, we propose a model of *governance artifact* to apply the governance decisions, taken at the tactic level, to the M2M infrastructure, using governance policies.

Such a governance artifact encapsulates an M2M entity of the infrastructure in order to provide a governance abstraction to the agents. Due to this abstraction, the agents can monitor and control the M2M infrastructure.

Monitoring is performed by the means of infrastructure *percepts*. These percepts are notified to the agents by the means of governance *events* and *properties*. Based on such an information, the agents can control the M2M entities using governance *operations* which trigger the artifact's infrastructure *controllers*.

In particular, governance artifacts embed *governance policies* which provide a means to regulate the behavior of an M2M M2M entity. Further more, such policies can be activated, tuned and deactivated by the agents in order to suit to the governance needs and to the infrastructure dynamics. Such policies automate some simple control loops using the artifact's infrastructure percepts and controllers.

In addition, the agents can *link* artifacts to each other, so the artifacts can coordinate together while running the governance policies.

In this chapter, we provide a detail description of such a governance artifact's design and mechanisms.

We first define the M2M governance policies expressing basic actions for monitoring and adapting the M2M infrastructure functioning. Then, we describe the general architecture of artifacts that interpret such governance policies to monitor the M2M infrastructure. Finally, we present three different type of artifacts that we have defined according to the governed M2M infrastructure entities.

8.1 M2M Governance Policies

M2M governance policies control the behavior of a single entity of the overall M2M infrastructure. They regulate such an entity's behavior to make it achieve a certain objective. These policies are configured and set up by the governance agents' tactics.

In this section, we define the language used to express these policies (Section 8.1.1). We give three examples in relation to three parts of the M2M infrastructure: (i) a device management policy, in Section 8.1.2, (ii) an application management policy, in Section 8.1.3, and (iii) a Telco management policy, in Section 8.1.4.

8.1.1 Governance Policy Language

A **Governance Policy** is a *"parametrized reactive regulation behavior"*. That is, a policy reacts to *events* occurring on the governed M2M system that do not fit with the expected behavior and provide *actions* to regulate such a behavior.

The language used to define such policies is based on Event-Condition-Action (ECA) rules in order to offer an *high-level declarative syntax*.

As stated in the State-of-the-Art (see Chapter 4), a high-level declarative approach helps to achieve a better objective definition. Thus, the governance policy language should express the expected (or unexpected) behavior of the governed M2M entity and the means to repair such a behavior in a simple manner, so that it can be easily parametrized.

In addition, ECA provide means for a simple and fast decision process. Contrary to governance tactic decisions, governance policy decisions should be fast enough to react to unintended events and directly correct them. Hence, the policy decision process should be simple enough, yet covering a wide range of possibilities, to regulate the M2M infrastructure.

ECA Rules are used to define high-level declarative reactive behaviors. The general syntax of an ECA rule is a triplet $\langle \varepsilon; \chi; \alpha \rangle$, where ε is an *event*, χ is a *condition*, and α is a set of actions. It is evaluated by the ECA engine as follows:

$$\textsf{when } \varepsilon \textsf{ if } \chi \textsf{ do } \alpha$$

That is, such a rule is triggered when the event ε occurs, to perform the actions α, provided that the condition χ holds.

ECA-based Governance Policy. A governance policy drives the M2M infrastructure's entities behavior by determining which actions to perform, depending on the context, when unintended events are observed.

Thus, each policy is defined by a set of policy rules $\mathcal{R}_p$ that determines the policy actions to perform. Each policy rule $r_p \in \mathcal{R}_p$ is specified by ECA rules denoted by a triplet $r_p :: \langle \varepsilon_p; \chi_p; \alpha_p \rangle$.

The language for specifying such ECA rules is based on: (i) a policy event ε_p, (ii) a policy condition χ_p, and (iii) a policy action α_p. The syntax for such governance policy elements is described in Table 8.1.

Policy Events. A governance policy can be triggered when a policy event occurs. Such an event can consist of (i) an M2M event, (ii) a policy event, or (iii) a tactic event.

Policy Conditions. The governance policy determines the actions to perform depending on the policy condition. Such a condition is expressed in terms of (i) the policy event's properties, (ii) the internal properties, (iii) the (sub-)requirements monitored locally, and (iv) the policy parameters set up by the agents. Such conditions can be combined using boolean unary and binary operations.

Policy Actions. The policy actions are actions to be performed by the governance policy engine (see Section 8.2.7). We consider four types of policy actions: (i) alerts to notify the agents, (ii) internal actions to be performed by the policy manager, (iii) delegation of an action to another policy manager, and (iv) M2M actions that should be performed on the M2M infrastructure. Such actions can be combined using a sequence operator. We chose to consider this single operator in order to guarantee a better consistency of the M2M policies.

Using such a language, we can declare M2M governance policies reacting to events occurring on the monitored infrastructure. In the next three section, we give examples of such policies for governing *devices' battery consumption* (Section 8.1.2), *client applications' activity* (Section 8.1.3) and *balancing workload among core platforms* (Section 8.1.4).

	Type	*Syntax*	*Description*
Events	M2M Events	`m2m(E)`	Event occurring at the M2M infrastructure level
	Policy Event	`policy(E)`	Event generated by another governance policy
	Tactic Event	`tactic(E)`	Event triggered by a governance tactic agent
Conditions	Event Property	`E.p`	Property held by the triggering policy event
	Internal Property	`prop(P)`	Property defined and maintained internally
	Requirement	`req(R)`	State of a requirement monitored locally
	Policy Parameter	`param(P)`	A parameter set at the governance tactic layer
	Unary Operation	`Op(C)`	Unary operation on a policy condition
	Binary Operation	`(C Op C)`	Binary operation on two policy conditions
	Void	-	No condition –i.e. always true
Actions	Alert	`alert(A)`	Alert to notify to the governance tactic layer
	Internal	`do(A)`	Perform an internal action `A`
	Delegation	`delegate(P,A)`	Delegate an action `A` to a policy manager `P`
	M2M Action	`perform(A)`	Action performed on the M2M infrastructure
	Action Sequence	`(A [; A]*)`	Several policy actions to be performed in sequence

Table 8.1: Syntax Elements of an M2M Governance Policy.

8.1.2 Device Lifetime Management Policy

As we have identified the M2M device lifetime constraint as a major issue in M2M, we define a governance policy to tackle such an issue. The *Device Lifetime Management* policy aims at ensuring the consistency of a device group by splitting the critical M2M devices to another device group.

In fact, all the M2M devices of a device group must provide the same services with the same level of QoS, agreed as a SLA. Hence, splitting the device group allow to define new a SLA between the critical devices and the M2M client application, whilst ensuring the satisfaction of the SLA for the other M2M devices.

Such a governance policy is expressed by four rules, reported in Table 8.2. The policy is parametrized with a minimum number of M2M devices necessary to ensure the consistency of the device group.

Rule 1: The event driving the policy is triggered by monitoring the M2M infrastructure for detecting critical lifetime state of M2M devices. Such an event is always notified to the governance tactic agents.

Rule 2: When the number of critical M2M devices is sufficient to create a new device group, another rule is triggered to create a so-called “degraded device group”. This new device group is associated to the current as it belongs to the same stakeholder, and it is composed of the same type of M2M devices.

Rule 3: If a degraded device group is set, the critical devices are directly moved to this group. The last action in the second rule, allows the policy engine to re-evaluate the event, considering a new context, and so, to be evaluated by the third rule.

```
(1) E: m2m(criticalDevicesDetected)
    C: -
    A: alert(criticalDevicesDetected(criticalDevicesDetected.devList))
-----------------------------------------------------------------------------
(2) E: m2m(criticalDevicesDetected)
    C: ((criticalDevicesDetected.nbDevices > param(pDevGpConsistency))
        and not(param(link.degradedDeviceGroup).setup))
    A: perform(createDegradedDeviceGroup) ;
       alert(degradedDevicesGroupCreated(param(link.degradedDeviceGroup))) ;
       do(m2m(criticalDevicesDetected))
-----------------------------------------------------------------------------
(3) E: m2m(criticalDevicesDetected)
    C: param(link.degradedDeviceGroup).setup ;
    A: permform(migrateDevices(criticalDevicesDetected.devList),
                               param(link.degradedDeviceGroup)) ;
       alert(devicesMigrated(criticalDevicesDetected.devList,
                             param(link.degradedDeviceGroup)))
-----------------------------------------------------------------------------
(4) E: policy(changedDeviceNb)
    C: ((policy(changedDeviceNb).nb < param(DevGpConsistency))
        and not(param(link.degradedDeviceGroup).setup))
    A: permform(migrateDevices(policy(deviceList).all),
                               param(link.degradedDeviceGroup)) ;
       alert(devicesMigrated(policy(deviceList).all,
                             param(link.degradedDeviceGroup))) ;
       alert(emptyDeviceGroup)
```

Table 8.2: Device Lifetime Management M2M Governance Policy Rules.

Rule 4: In order to keep the first device group consistent, we move all the remaining M2M devices to the degraded device group, when the number of safe devices becomes lower than the consistency parameter.

8.1.3 Application Message Threshold Policy

In order to enforce the M2M client application to comply with the SLA, we define a governance policy to control the number of messages it sends to the M2M devices. The *Application Message Threshold* policy aims at controlling each message sent by the client application and assigning a threshold above which the messages will be blocked.

Such a governance policy is expressed by three rules, reported in Table 8.3. It uses the SLA value *[command_rate(R)]* in order to determine the number *R* of messages the application has the right to send

```
(1) E: m2m(app_sending(M))
    C: app_sending(M).freq ≤ R
    A: perform(send_message(M)) ;
       alert(authorized_message(M.dev, app_sending(M).freq))
(2) E: m2m(app_sending(M))
    C: ( M.priority = URGENT
         and (app_sending(M).freq ≤ R)
    A: perform(send_message(M)) ;
       alert(rq_failure(msg_freq(M.app,app_sending(M).freq), tolerated))
(3) E: m2m(app_sending(M))
    C: (M.priority ≠ URGENT and app_sending(M).freq > R)
        or (M.priority = URGENT and app_sending(M).freq > R + T)
    A: perform(abort_sending(M)) ;
       alert(rq_violation(msg_freq(M.app,app_sending(M).freq), blocked))
```

Table 8.3: Application Message Threshold M2M Governance Policy Rules.

to the M2M device group. In order to allow exceptional messages, a tolerance value *T* can be parametrized by the governance tactic agents.

The event driving this policy is triggered by monitoring the M2M infrastructure for messages sent by the client application to the devices. We consider only a single vertical contract SLA per instance of the policy.

Rule 1: Messages are authorized to be sent if the client application complies with the *[command_rate(R)]* SLA attribute, and agents are notified.

Rule 2: In case the message is urgent and does not overpass the tolerance parameter, we allow the message to be transmitted. Then an alert is sent to the governance tactic agents specifying the message was tolerated to be sent while violating the SLA.

Rule 3: In all other cases, the messages are directly aborted and the agents are notified the client application violated the SLA.

8.1.4 Horizontal Load Balancing Policy

To enforce the resilience of the Telco infrastructure's compliance with the horizontal requirements, we define a governance policy to manage load balancing between several instances of M2M core platforms. The *Horizontal Load Balancing* policy is based on a simple greedy algorithm to distribute the core platform's functionalities between the different instances.

Such a policy is expressed by five rules, reported in Table 8.4. The policy is parametrized with available instances of M2M core platforms providing alternative to the component's functionalities, an overload tolerance threshold and a critical level for each functionality.

Policy rules 1, 2, 3 and 4 are triggered when the monitored platform is overloaded.

Rule 1: If an alternative component is available –i.e. another instance exists and is not overloaded– the call to the component is redirected to the alternative one.

Rule 2: Otherwise, if the overload does not exceed the tolerance threshold, the policy let the component handle the invocation with some delays.

Rule 3: If the overload is bigger than the tolerance threshold and no alternative component is available, the policy will abort the call unless the task is marked as critical.

Rule 4: If the task is critical, the component invocation is removed from the platform and kept until the component or one of its alternative is free.

Rule 5: When a platform is available, it delegates to it the pending invocations.

8.2 Governance Artifact Architecture

Artifact provides an abstraction of the deployed M2M infrastructure to the agents and a policy-based control loop so that the agents can focus on the governance tactic reasoning.

Figure 8.1 provides a representation of such a governance artifact. It is composed of (i) *governance properties* and (ii) *events* that notify the agents of the M2M infrastructure's state, (iii) *governance operations* that allow the agents to act upon the M2M infrastructure's behavior, (iv) *links* that enable governance artifacts to coordinate together, (v) *governance monitoring* and (vi) *control* modules which monitor and act upon the M2M infrastructure, and (vii) a *policy engine* that enact the governance policies set up by the governance agents.

The three first modules are driven by the governance tactic agents' needs, while the others are defined according to the M2M infrastructure specifications.

In this section, we describe the governance artifact architecture that we propose in order to govern the M2M infrastructure. Each section describes the different modules of such a governance artifact.

8.2.1 Governance Properties

The governance properties are artifact observable properties. When an agent observes the artifact, these properties are added to its beliefs base. Hence, we use this to let the agents know about the current state of the M2M governance.

We can identify three types of governance properties: (i) M2M entity information, (ii) requirements fulfillment, and (iii) regulation policies performances.

```
(1) E: m2m(overload(O))
    C: param(alternatives(O.component).available
    A: perform(delegate(alternatives(O.component).first_alt,
                        O.invocation))
       alert(rq_failure(overload(O.component, O.load), delegated))
--------------------------------------------------------------------
(2) E: m2m(overload(O))
    C: (not (param(alternatives(O.component).available))
        and (O.load ≤ O.capacity + param(tolerance))
    A: perform(wait_and_execute(O.invocation))
       alert(rq_failure(overload(O.component, O), delayed))
--------------------------------------------------------------------
(3) E: m2m(overload(O))
    C: ((not (param(alternatives(O.component).available))
          and (O.load > O.capacity + param(tolerance))
        and (not param(is_critical(O.component))))
    A: perform(abort_invocation(O.invocation))
       alert(rq_violation(overload(O.component, O), aborted))
--------------------------------------------------------------------
(4) E: m2m(overload(O))
    C: ((not (param(alternatives(O.component).available))
          and (O.load > O.capacity + param(tolerance))
        and param(is_critical(O.component)))
    A: do(hold_invocation(O.invocation))
       alert(rq_violation(overload(O.component, O), pending))
--------------------------------------------------------------------
(5) E: policy(available_instance(I))
    C: prop(pending_invocations.count > 0)
    A: delegate(I, pending_invocations)
```

Table 8.4: Horizontal Load Balancing M2M Governance Policy Rules.

M2M Entity Information. With such properties, the governance artifact let the agents know about the M2M entity they govern. It provides information such as their identifiers within the M2M infrastructure, their individual objectives and their activity.

This allows the agents to take into account the entities' state in their governance decisions. We describe such governance properties that are common to all the governance artifacts.

- `m2m_entity(Type(EID),SID)` denotes the M2M entity governed by the artifact, where `EID` is the entity's identifier, and `SID` is the owner of the entity (i.e. stakeholder identifier). `Type` is the nature of the M2M entity, which can be one of the following values: `application`, `platform`, `gateway`, `repeater`, `actuator`, `sensor`.
- `subscriptions(EID,CID)` denotes the subscriptions the `EID` M2M entity is involved in, `CID` refers

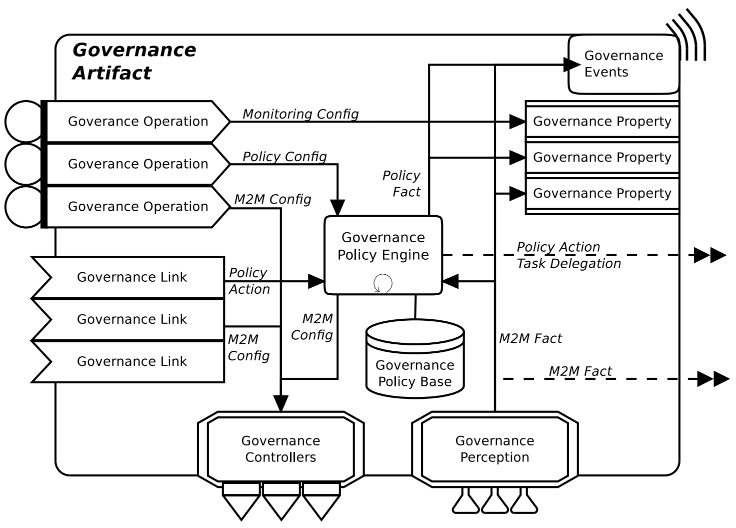


Figure 8.1: Architecture of a Governance Artifact.

to the corresponding vertical contract.

- `entity_state(EID,State)` denotes the state of the `EID` M2M entity, where `State` can be the entity requirements' satisfaction (`success_rate(R)`) or related to each M2M entities individual objectives (detailed in Section 8.3).

Requirements Fulfillment. A governance artifact allows to monitor partial requirement fulfillment. For each of them, it provides the local functionality monitored that partially fulfills the requirement. It also provides data about their fulfillment, the actual tolerance rate or failures, enabling the agents to keep track of the causes of failures.

This allows the agents to take into account the M2M infrastructure activity to reason about the governance strategy.

- `requirements_local_functionalities(RQ,F)` denotes the functionalities `F`, governed locally, that fulfill or contribute to fulfill the requirement `RQ`.
- `requirements_local_config(RQ,qos(QoSParam,QoSValue))` denotes the QoS values `qos(QoSParam,QoSValue)` measuring the `RQ` requirement fulfillment.
- `requirements_tolerance(RQ,T)` denotes the tolerance rate `T` for the `RQ` requirement fulfillment.
- `requirements_status_history(RQ,Date,Status)` traces the requirements' fulfillment and failures denoted by `Status` which values can be described by `fulfilled(F[,QoS])` or `failed(F,Cause[,QoS])`.

Regulation Policies Performances. Finally, the artifact allows the agent to evaluate the performances of the regulation policies. It provides data about the actual the policy parameters, the activation of the policy rules and their result.

This allows the agents to evaluate the governance policies they have set up in order to tune them or to change their tactic.

- `policies(PolicyID,{Available|Activated}[,Params])` denotes the governance policies available with their activation status and the parameters set up by the governance tactic agents.
- `policies_activity(PolicyID,Date,Activity)` traces the activity of the governance policy. Such an activity can be denoted by the following terms : `triggered(E,C,A)`, `enforced(OldParam,NewParam)`, and `relaxed(OP,NP)`.
- `policies_stats(PolicyID,ActivFreq,SuccessRate,EnforceRate,RelaxRate)` provides performance statistics concerning the governance policies.

These properties allows the governance tactic agents to evaluate the governance functioning, and so, to reason about its application and feasibility. Additionally to such information, governance events are used to alert the agents about critical situations so that they can focus more on the governance failures.

8.2.2 Governance Events

The governance events are artifact signals that are sent to the agents to alert them. Thus, they are used to notify the agents about the governance problems to be treated quickly. The governance events can be (i) infrastructure events, (ii) requirement failures, or (iii) policy alerts.

Infrastructure Events. When the M2M infrastructure is modified, the artifact notifies the agents about these modifications. Hence, the agents can be aware of the registration of a new stakeholder to govern, the stakeholders departure and the introduction of a new vertical subscription.

- `new_entity_registered(Type(EID), SID)` notifies the governance tactic agents that a new M2M entity is registered in the M2M infrastructure.
- `entity_unregistered(Type(EID), SID)` notifies the governance tactic agents that an M2M entity was withdrawn from the M2M infrastructure.
- `subscription_requested(SuID,EIDs,SLA)` notifies the governance tactic agents that a vertical subscription, identified by `SuID`, is submitted. The subscription proposal involves the list of M2M entities denoted by `EIDs`, and is specified by the `SLA` parameter.

Requirement Failures. The agents can be notified of partial requirement failures thanks to this type of events. When a requirement is failed, an alert is triggered. Such an alert can be different whether the failure tolerable –i.e. requirement failure– or not –i.e. requirement violation.

- `rq_failure(RQ(Cause)[,Action])` notifies a tolerable requirement failure, its cause and, potentially, the policy action used.
- `rq_violation(RQ(Cause)[,Action])` notifies an important requirement failure, its cause and, potentially, the policy action used.

Policy Alerts. Finally, alerts are generated when the governance policies are inaccurate. This can be the case when a failure is persists despite the policy actions or if the policy is not used during a while.

- `failing_policy(PolicyID,Cause)` notifies the agents that the policy `PolicyID` fails to make the M2M infrastructure match the requirements.
- `under_used_policy(PolicyID,LastUsed,AvgUsage)` notifies the agents of a useless activated policy.

These events allows the artifact to alert the agents about governance and infrastructure issues so that they handle the problems more urgently. Typically, it is expected that the agents react to these signals setting up adequate actions thanks to the governance operations provided by the artifact.

8.2.3 Governance Operations

The governance operations are artifact operations that agents can use to act upon the artifact. Thus, they are used to help the agent to configure and refine the governance policy –i.e. the artifact's behavior. Such operations can be used (i) to configure the M2M infrastructure, (ii) to configure the local requirement monitoring, and (iii) to configure the governance policies.

M2M Infrastructure Configuration. These operations allows to modify the M2M infrastructure's configuration and to parametrize its configuration.

- `deploy_entity(in:T, in:SID, out:EID[, in:R])` deploys a new instance of an M2M entity of type `T` –e.g. a core platform, a component, an M2M gateway. . . – belonging to the stakeholder denoted by `SID`. The M2M entity instance's identifier is returned by the output parameter `EID`. In case of a physical deployment –i.e. an M2M gateway or repeaters– the action can consist of a simple recommendation, specified by `R`, to the corresponding stakeholder administrator, so that he might perform the deployment further.
- `dispose_entity(in:EID[,R])` terminates and free the M2M entity instance denoted by `EID`. This should rarely be used for physical M2M components due to the human intervention expensive cost.
- `deploy_subscription(in:SuID,in:QoSParams)` configures the M2M entity governed by the artifact to handle the subscription denoted by `SuID` with the QoS parameters specified by `QoSParams`.
- `terminate_subscription(in:SuID)` configures the M2M entity governed by the artifact to stop handling the subscription denoted by `SuID`.

- `compensate(in:StID,in:Amount)` operate a financial compensation of `Amount` between entity's stakeholder the stakeholder identified by `StID`.

Local Requirements Configuration. The governance policy artifact controls the –potentially partial– requirements fulfillment of the M2M entity they govern. So, we define operations to specify how such requirements should be treated, by specifying the corresponding tolerance threshold.

- `eval_rq_feasibility(in:RQSet,out:Feasibility[,out:RQProposal])` evaluates the feasibility of the `RQSet` requirement set, based on a local heuristic. The output parameter `Feasibility` denotes such feasibility as a boolean value. If the proposal is not feasible, the operation can potentially provide another feasible requirement specification (`RQProposal`).

- `enforce_rq_tolerance(in:RQ_ID[,in:ToleranceStep],out:Tolerance)` enforces the constraints on the requirement denoted by `RQ_ID`. The agents can specify the enforcement step with `ToleranceStep`. The new tolerance threshold is returned by the output parameter `Tolerance`.

- `relax_rq_tolerance(in:RQ_ID[,in:ToleranceStep],out:Tolerance)` relaxes the constraints on the requirement denoted by `RQ_ID`. The agents can specify the enforcement step with `ToleranceStep`. The new tolerance threshold is returned by the output parameter `Tolerance`.

Governance Policy Configuration. In order to enforce the M2M infrastructure requirement fulfillment, we define policy configuration operations that allow the agents to set up and configure governance policies to be applied on the governed M2M entity by the governance artifact.

- `activate_policy(in:Policy[,in:Policy_Params],out:PID)` starts the policy denoted by the `Policy` parameter to regulate the governed M2M entity, with the parameters given by `Policy_Param`. An identifier for the corresponding instance is returned by the output parameter `PID`.

- `deactivate_policy(in:PID)` terminates the governance policy instance denoted by `PID`.

- `configure_policy(in:PID,in:Param,in:ParamV)` configures the `PID` policy instance's `Param` parameter with the `ParamV` value.

- `enforce_policy(in:PID,in:Param[,Step],out:ParamV)` enforces the `PID` policy instance's `Param` parameter, with a potential value `Step`. The new parameter value is returned by the output parameter `ParamV`.

- `relax_policy(in:PID,in:Param[,Step],out:ParamV)` relaxes the `PID` policy instance's `Param` parameter, with a potential value `Step`. The new parameter value is returned by the output parameter `ParamV`.

8.2.4 Governance Links

The governance links are artifact linkable operations that allows other artifacts to use them. Thus, they are used to enable coordination between the governance artifacts.

This requires the reference to the other artifacts to be specified to the former one. Such a connection is operated by the governance agents. As the reference might differs from a type of artifact to another the specific link references will be described in Section 8.3.

Such governance links enable the artifact to (i) propagate M2M states, (ii) coordinate several policies, and (iii) delegate governance tasks.

M2M State Propagation. Similarly, changes on an M2M entity might impact the governance of other M2M entities.

- `m2m_state(in:EID,in:State,in:StateV)` notifies the linked artifact that the M2M `EID` entity's state –denoted by `State`– is now of value `StateV`. As an example, such a state could be the M2M entity's workload or availability.

Policy Coordination. When governance policy requires several artifacts –i.e. several M2M entities governed by different artifacts– modifications to such a policy configuration should be propagated to all the linked artifacts.

- `update_policy(in:PID,in:Param,in:ParamV)` updates the linked artifact `PID` policy's parameter's (denoted by `Param`) value has been changed to `ParamV`.

Governance Task Delegation. Finally, we define a link operation to distribute governance tasks to other artifacts.

- `delegate(in:Cmd[,in:Params][,out:Result])` delegates a task, denoted by `Cmd`, and the possible parameters `Params`, to the linked artifact. Such a task can be treated either by the policy engine (see Section 8.2.7) or directly by the M2M controller (see Section 8.2.6). If appropriate, the result can be returned by the output parameter `Result`.

8.2.5 Governance Perception

The governance perception is a set of sensors that allow the artifact to perceive the M2M infrastructure events. These M2M events are treated by the perception module to feed the other governance modules of the artifacts.

The module monitors the M2M infrastructure's performance status and the functionalities being used. Then, it treats these data to compute M2M events that are dispatched.

Modules / M2M Events	*Properties*	*Events*	*Policy Engine*	*Links*
`entity_state`	×	×	×	×
`invocation`			×	×
`subscription_proposal`		×		
`subscription_registered`	×	×		
`message`		×	×	

Table 8.5: M2M Events Dispatch to the Artifacts' Governance Modules

M2M Events. The perceived events are those providing accurate information with respect to the governance concerns. Based on such event, the governance perception generates the following M2M events:

- `entity_state(EID,Status)` provides the status denoted by `Status` of the `EID`. Such a status addresses the workload, the availability, a requirement fulfilment, the deployment, the registration or the termination of the corresponding M2M entity.
- `invocation(EID,F,Params)` means the M2M entity `EID` is being invoked to execute the functionality `F` with the list of parameters denoted by `Params`.
- `subscription_proposal(SuID,EIDs,SuType[,SuParam])` denotes a subscription proposal –i.e. to be validated– requiring the involvement of the stakeholders' entities defined by the list `EID`. The type of subscription is specified by `SuType` and, potentially, by a list of SLA and compensation parameters.
- `subscription_registered(SuID,EIDs,SuType[,SuParam])` denotes a registered subscription –i.e. already validated– involving the stakeholders' entities defined by the list `EID`. The type of subscription is specified by `SuType` and, potentially, by a list of SLA and compensation parameters.
- `message(SenderEID,ReceiverEID,MData[,Urgency][,QoSData])` notifies that a message, sent by the `SenderEID` M2M entity, is being transmitted to `ReceiverEID`, containing the data `MData`. Such message may be qualified by a level of urgency defined by `Urgency` and QoS data denoted by the list `QoSData`.

Event Dispatch. Once the perception module has computed the M2M events, they are transmitted to the other modules of the artifact to be treated. A publish/subscribe mechanism is used to feed the different governance modules.

Hence, the M2M events are dispatched according to the different modules needs: (i) the *Governance Properties* module receives information relevant for updating long term/persistent governance properties, (ii) the *Governance Events* module receives information that may impact the governance tactic, (iii) the *Governance Policy Engine* module receive information to be treated by the running policies, and (iv) other artifacts can be notified using their *Governance Link* modules so that the state of the M2M infrastructure can be propagated. Such a dispatch is synthesized in Table 8.5.

8.2.6 Governance Controller

The governance controller allows the artifact to control the M2M infrastructure's behavior. Such a controller is used by the other governance modules: (i) the governance operations (invoked by the agents), (ii) the artifact links (invoked by another artifact), and (iii) the policy engine (triggered by the policies).

To do so, we define a set of commands specifying actions the controller can perform upon the governed M2M entity.

- `legacy(in:F)` lets the legacy functionality `F` being executed without notifications. This is the default behavior.
- `notify(in:F)` notifies the usage of a functionality `F` to the other governance modules.
- `intercept(in:F)` intercepts the `F` functionality calls and notifies the artifact's governance modules to treat it (e.g. Policy Engine).
- `redirect_calls(in:F,in:F')` redirects the `F` functionality calls to another instance `F'` (e.g. Governance Link).
- `invoke(in:F,in:Params)` invokes a functionality `F` on the legacy system. `Params` is the list of parameter values used to invoke the functionality.
- `compensate(in:StID,in:Amount)` charges the artifact's governed entity's stakeholder and credits the stakeholder identified by `StID` by the amount defined by `Amount`.

8.2.7 Governance Policy Engine

The governance policy engine enacts the policies activated by the agents by executing the corresponding set of policy rules, as defined in Section 8.1. Artifacts embed and enact such governance policies as follows:

- The governance policies that can be used by the engine are contained in a *Governance Policy Base*. The policies are configured, started and stopped by the agents through the policy-related *operations*.
- Then, the policy engine evaluates the rules when (i) M2M Events are perceived by the governance perception, and/or (ii) linked operations are invoked by other artifacts. The policy engine embeds an ECA evaluation engine that analyzes the event and the current conditions to define one or several governance actions.
- Finally, the *actions* resulting from the policy rules evaluation are executed either by (i) the policy engine for governance properties and events updates, (ii) the governance controller for M2M actions, or (iii) the by an external artifact's linked operation for delegation actions.

Hence, the *Governance Policy Engine* is a central module of the governance artifact. It makes the governance policies decided by the governance tactic layer effective, thanks to the combination of all the artifact's governance modules.

8.3 M2M Governance Artifacts

As identified in Chapter 7, the M2M governance addresses different concerns depending on the different stakeholders (see Section 7.1.4). The different types of agents have different needs of M2M abstraction. Therefore, in this section, we define the governance artifacts from the different abstraction needs.

We have defined three type of artifacts addressing the different governance policy concerns: (i) the *DeviceArt* artifact, (ii) the *ApplicationArt* artifact, and (iii) the *TelcoArt* artifact. All the governance artifacts share the same base architecture, described in Section 8.2. They provide specifics governance modules related to the concerns they address.

In the remainder of this section, we present the specifications of such governance artifacts, namely: the device artifact (*DeviceArt*) in Section 8.3.1, the application artifact (*ApplicationArt*) in Section 8.3.2, and the Telco infrastructure artifact (*TelcoArt*) in Section 8.3.3.

For each of them, we specifies only the additional contents to the base architecture, presented previously. Before going further, let us note the governance policy engine is the same for all the artifacts, but their Governance Policy Base contains a different set of policies.

8.3.1 Device Artifact

The device artifact (*DeviceArt*) is governance abstraction artifact to manage the M2M devices' performance and life-cycle. A *DeviceArt* artifact represents a *device group* M2M entity.

An M2M device group is a set of M2M devices which (i) belongs to the same stakeholder, (ii) is located in the same consistent geographic area, and (iii) which provides the same services.

Such a device group is identified by its M2M entity identifier (`EID`). Hence, the whole device group is committed to the same vertical contracts as a block. As a consequence, all the group's devices share the same vertical requirements, and so, they must maintain a similar life-time viability. To handle this, we allow critical devices to be migrated to a degraded device group, that is, a group that share the same specifications as the previous one, except its vertical commitments. Thus, we can define lower SLA so that the device life-time last longer.

Such a governance artifact helps the governance tactic agents for (i) the route maintenance and selection, (ii) the M2M device monitoring and configuration, and (iii) the device group management.

Thus, the *DeviceArt* artifact provides M2M device-specific governance properties, events, operations, links, and controller modules. Let us note that we do not need additional perception specifications. In addition, we provided an example of device-specific governance policy in Section 8.1.2.

Device Properties. In this section, we detail the M2M device-specific governance properties. First, we define the properties device group, then we detail the state of an M2M device group entity to, finally, specify the group's routing and service properties.

Device Group. As mentioned previously, the device group is the unit for the governance abstraction. Thus, we define device group-related properties to guide the agent's governance tactics.

- `device_group(EID,StID,Devices,Type)`: the governed M2M device group, where `EID` refers to the device group's identifier, `StID` is the devices' owner stakeholder identifier, `Devices` denotes the list of M2M device the group is composed of, and `Type` defines the type of devices –i.e. `sensor`, `actuator`, or `repeater`.
- `device_group_lifetime(EID,DD,ELT,VT)`: the stakeholder's life-time requirement, where `EID` refers to the device group's identifier, `DD` is the devices' deployment date, `ELT` is the expected life-time, and `VT` is the devices' viability tolerance (with $0 < VT < 1$) regarding the expected life-time.
- `devices(DevID,EID,LT,V)`: state of the M2M devices composing the device group: `DevID` is the M2M device's identifier (e.g. typically its MAC address), `LT` is the measured actual lifetime of the device, and `V` is the resulting actual viability –computed as follows: $LT/(DD + ELT - Today)$.

Entity State. Such as every M2M entity, the governance artifact provides an `entity_state(EID,State)` property. As regards the M2M device groups, the `State` parameter is defined by the following values:

- `success_rate(R)`: the average success rate, with respect to the device group's fulfillment of all the vertical contract it is engaged in.
- `lifetime_heuristic(ExpL,ReqL)`: the expected group's average life-time (`ExpL`) according to an embedded heuristic, and the required life-time (`ReqL`) specified by the device group's owner.

Device Routing. In order to help the governance agents maintaining efficient routes for the device group, we define route specification and evaluation properties.

- `routes(EID,RouteID,route(GWID,Route[,Repeaters])[,QoS])` denotes the routes usable to reach the `EID` group's devices. Each route is identified by its `RouteID` identifier. The route is defined by the gateway to use reach the devices (`GWID`), a route within the WSAN (`Route`) consisting of either a pre-defined path expressed as a graph (`path(PathGraph)`) or a routing algorithm (`routing(Algo)`). The route may require extra-devices to act as repeaters for the device group, defined by a list of devices (`Repeaters`). Finally, the route parameter `QoS` may provide a list QoS-related information, such as the (theoretical or measured) average latency, data loss, life cost and viability.
- `selected_route(EID,RouteID)` denotes the route selected (`RouteID`) for the `EID` device group.

Device Services. To engage the device group in vertical contracts, it is necessary to indicate what services the devices provide for which performance.

- `services(EID,CMD,QoS)` describes the services provided by the device group `EID`. For each (sensing or actuating) service provided –denoted by `CMD`– a list of QoS performance is provided, such as the maximal sensing/actuating frequency `freq(F)`, the maximal message transmission rate `msg_rate(R)`, and the command (`cmd_weight(W)`) and response (`res_weight(W)`) messages' weight.
- `performance(EID,CMD,QoS)` indicates the services performance measured for the device group `EID`. This is computed by the artifact along the device group's life-time. The definition of `CMD` and `QoS` terms follows the one given previously.

Device Events. Related to the M2M device group properties, we now specify governance events to alert the tactic agents about device related critical situations. Such specific events are concerned with the device group life-cycle governance, and the routing governance.

Device Group Life-Cycle. In order to adapt the governance to the dynamics of the M2M device groups, we define alerts to notify of device group deployment and termination, introduction and departure of M2M devices, and devices' critical states.

- `new_devices_group(EID)`: a new device group is deployed with the identifier `EID`.
- `device_group_terminated(EID)`: the current device group (`EID`) no longer exists.
- `device_added(EID,DevID)`: a device (`DevID`) is added to the group.
- `device_removed(EID,DevID)`: a device (`DevID`) is removed from the group.
- `critical_device(EID,DevID,LT,V)`: a device (`DevID`) in the group has a low expected life-time `LT` and has reached a critical viability state `V`.

Device Group Routing. We define routing events that alert the tactic agents of the WSAN routes evolution. That is, route selection and new routes available so as deprecated routes.

- `route_selected(EID,RID)`: a new route (`RID`) was selected for the device group (`EID`).
- `new_route_available(EID,RID)` a new route (`RID`) is available for the device group (`EID`) –e.g. due to a deployment of repeaters, or to a software upgrade.
- `deprecated_route(EID,RID[,Selected])` a route (`RID`) is no longer available for the group (`EID`) –e.g. due to some devices' crash. In case the route was currently used (`Selected`), the agents should replace it quickly.

Device Operations. We now define governance operations that allow the tactic agents to act upon the M2M devices' configuration. In particular, such operations enable to configure the devices' activity so that they provide their services according to the vertical SLAs, but they also enable the agents to manage the device group life-cycle and routing.

Command	*Device Type*	*QoS Parameter*	*Description*
`MUTE`	*all*	–	Mute the device *[Default]*
`REPEAT`	*all*	–	Set the device on repeater mode
`NOTIFY`	*sensor*	`SensFreq` `NotifFreq`	Sense data at a `SensFreq` frequency, and send them at a `NotifFreq` frequency
`ONEVENT`	*sensor*	`MaxNotifFreq`	Send data when event happens with a maximum frequency of `MaxNotifFreq`
`ONDEMANDACT`	*actuator*	`ActACK`	Perform actuation when requested by applications and send acknowledgement if `ActACK` is true
`SCHEDULEDACT`	*actuator*	`ActFreq` `ActACK`	Perform actuation at `ActFreq` frequency and send acknowledgement if `ActACK` is true

Table 8.6: M2M devices configuration commands and QoS parameters.

Service Configuration Operations. We define an operation activate and deactivate the M2M devices' services. Such services may vary depending on the type of devices.

- `configure_devices(in:CMD,in:List<QoSParam>)`: configures the devices to activate the service denoted by `CMD` according to the requirements defined by the list of `QoSParams`, and sets up the requirement monitoring by the artifact. A `QoSParam` parameter is tuple `<QoSCriterion, QoSValue>`, where the `QoSCriterion` depends on the service activated. Table 8.6 provides the correspondence between the service commands and the QoS parameters.

Viability Operations. To manage the device group life-cycle, we define two operations: one enable to create a degraded device group and one to merge the two device groups when the current group is obsolete. Using the *Device Lifetime Management* policy, given in Section 8.1.2, the agents can ensure the M2M device will migrate to this degraded device group when they reach a critical state.

- `split_critical_devices(out:EID,out:List<DevID>)`: creates a new device group and migrates all the critical devices to the new group. It returns the degraded group identifier (`EID`) and the migrated devices list (`List<DevID>`) as output parameters.
- `merge_devices_group(out:EID)` moves all the current group's devices to the degraded device group, and terminates the current device group. The output parameter `EID` corresponds to the degraded device group identifier.

Routing Operations. Such operations allows the agents to select the most accurate route in order to ensure the M2M devices match the vertical SLAs.

- `select_route_max_viability(in:MaxLatency,in:MaxLoss,out:RouteID)`: selects the route –given as an output parameter (`RouteID`)– that maximize the devices viability and providing an acceptable QoS with respect to the `MaxLatency` and `MaxLoss` criteria.

- `select_route_lower_viability(in:RouteID,out:NewRoute)`: selects a route given as an output parameter (\verbNewRouteID+)– with a lower viability than the one given in parameter (`RouteID`) within the same device group.

- `find_external_routes(out:List<ExtRouteID>)`: explores other routes considering all the device groups, and returns it as an output parameter (`List<ExtRouteID>`). Let us note that such routes would need an agreement with the corresponding device groups' stakeholders to be used.

Device Link. Considering a device group M2M entity, we use an artifact link to connect it to the correspond degraded device group. Hence the tactic agents and the policy rules can refer to this link to manage the M2M device migration.

- `@LINK degraded_device_group(Degraded_EID)`: the M2M entity reference to the related degraded device group (`Degraded_EID`).

Device Control. We define extra control commands which enables to manage the device group life-cycle and configure the M2M devices' service and routing.

Device Group Life-Cycle. When creating and terminating the device groups or when migrating M2M devices from a group to another, it is necessary to register the device group's changes on the M2M core platform. The following commands allow the artifact to perform the corresponding actions:

- `create_device_group(out:EID)`: register a new device group on the M2M core platform.

- `close_device_group(out:EID)`: terminates a device group on the M2M core platform.

- `migrate_devices(in:List<DevID>,in:EID)`: move the devices given in the list `List<DevID>` to another device group (`EID`).

Device Configuration. To configure the M2M devices according to agents or the policy decisions, we define two device configuration commands:

- `configure_service(in:CMD,List:<QoSParams>)`: configures the group's M2M devices to perform the service denoted by `CMD` according to the QoS parameters list (`List<QoSParams>`).

- `configure_route(in:RouteID)`: configures the group's M2M devices to route their messages according to the route defined by `RouteID`.

8.3.2 Application Artifact

The application artifact (*ApplicationArt*) is a governance abstraction artifact to manage the M2M applications' subscriptions and rights. An *ApplicationArt* artifact represents a *client application* M2M entity. That is, an external service that uses the M2M core platform to access to M2M devices.

Such a governance artifact helps the governance tactic agents for (i) managing the application's subscriptions, and (ii) controlling its access to the M2M devices.

Thus, the *ApplicationArt* artifact provides M2M application-specific governance controller modules. Let us note that we do not need to define additional properties, events, operations, links and perception modules. In addition, we provided an example of application-specific governance policy in Section 8.1.3.

Application Controls. We define extra commands which enables the artifact to configure the M2M infrastructure according to the vertical SLAs.

Subscription Controls. As the M2M application initiates the vertical contracts, we define two commands to start and terminate the corresponding subscriptions:

- `create_subscription(App_EID,Dev_EID,CID,SLA)`: registers a subscription identified by the contract identifier `CID`, between the application `App_EID` and the device group `Dev_EID`, with the SLA specified by SLA.
- `terminate_subscription(CID)`: unregisters the subscription identified by the contract identifier `CID`.

Application Rights. To control the client application's rights, we define two commands which enables the artifact to control the commands the application sends:

- `allow_app_cmd(Dev_EID)`: enables the M2M core platform to transmit commands the governed application sends to the device group `Dev_EID`.
- `block_app_cmd(Dev_EID)`: disables the M2M core platform to transmit commands the governed application sends to the device group `Dev_EID`.

8.3.3 Telco Infrastructure Artifacts

The Telco infrastructure artifact (*TelcoArt*) is a governance abstraction artifact to manage the M2M core platform and gateways. A *TelcoArt* artifact governs a *Telco component*, that is a component that provides one or several infrastructure functionalities.

Such a governance artifact help the tactic agents to ensure the Telco infrastructure provides a correct end-to-end connectivity between the M2M applications and the M2M devices. Hence, it (i) monitors a single component's functioning, and (ii) manage alternative components to ensure the resilience of the governed functionality.

In Section 8.1.4, we have defined an *horizontal load balancing* policy as an example of such an infrastructure management. We now define additional governance properties, events, operation, links and percepts and controls to unable the governance of such Telco components deployment and load management.

Telco Properties. We define governance properties, to allow the agents to know about the Telco component used to fulfill the functionalities and the possible alternative to such a component.

- `telco_component(EID,ComponentID[,ComponentDescr])`: component used (`ComponentID`) by the entity (`EID`), governed by the artifact, to implement and fulfill its functionalities. `ComponentDescr` can be used to provide additional qualitative information to guide the agents' choice, such the component's workload tolerance, its profit threshold, the algorithm or type of database used, for example.

- `available_alternatives(EID,ComponentID,AltComponentID[,AltDescr])`: alternative component (`AltComponentID`) to the one governed by the artifact (`EID`, `ComponentID`) and its description (`AltDescr`).

Telco Events. Such events are used to alert the agents about the governed component's workload. This allows the agent to analyze the Telco infrastructure usage, to plan further adaptations.

- `overloaded_component(ComponentID,Load)`: notifies an overloaded component (`ComponentID`), that is its workload (`Load`) overpass its tolerance.

- `underused_component(ComponentID,Load)`: notifies an underused component (`ComponentID`), that is its workload (`Load`) is too low regarding its cost.

- `redirected_invocation(F,AltComponentID)`: notifies a functionality invocation (`F`) that was redirected to an alternative component (`AltComponentID`).

Telco Operations.

- `deploy_alt_component(out:AltComponentID)`: deploys a new instance of the governed component's to to duplicate it as an alternative component.

- `distribute_functionality(in:F,in:AltComponentID)`: distributes the functionality `F` with the linked alternative deployed instance `AltComponentID`. This can be used in conjunction with a governance policy, such as the one given in Section 8.1.4.

- `shutdown_component(out:Successful)`: terminates the governed component instance to free resources. The output parameter `Successful` denotes the success of the operation as a boolean value.

Telco Links.

- `@LINK alt_instances(List<AltComponentID>)`: list of alternative components instances linked to the current artifact. Such references can be used to invoke the following linked operation.

- `handle_redirected_invocation(in:FInvocation[,out:ReturnV])`: redirects a functionality invocation (`FInvocation`) to the component governed by the linked artifact. The output parameter `ReturnV` can be used to retrieve the value returned by such an invocation.

Telco Percepts. The following additional governance percepts allows the artifact to handle the governed component's invocations, its workload and its deployment status.

- `invocation(FInvocation,IsBlocked)`: one of the governed component's functionalities is invoked, `FInvocation` denotes the invoked functionality with the given parameters, and `IsBlocked` is a boolean value stating whether the invocation is being performed by the governed component or if it is blocked.

- `component_load(Load)`: the actual governed component workload (`Load`).

- `component_deployment(EID,Deployment)`: the current governed component deployment state (`Deployment`), which can be `deployed` or `terminated`, on the `EID` M2M Telco infrastructure entity.

Telco Controls. Finally, we define additional governance controls to treat the component's invocation and deployment.

- `invoke_functionality(in:FInvocation[,out:ReturnV])`: invokes the governed component's functionality with some parameters, as defined by `FInvocation`, and potentially retrieves the value returned (`ReturnV`).

- `intercept_functionality_calls(in:F,in:Blocking)`: intercepts the component's functionality `F` invocations, blocking them depending on the `Blocking` parameter value.

- `redirect_calls(in:F,in:AltComponentID)`: redirects the component's functionality `F` invocations to another alternative component instance (`AltComponentID`).

- `deploy_component(in:EID)`: tries to deploy a component instance on the M2M Telco infrastructure entity (`EID`). The result is notified by `the component_deployment(EID,Deployment)` precept.

- `terminate_component()`: tries to terminate the governed component instance. The result is notified by `the component_deployment(EID,Deployment)` precept.

8.4 Synthesis

In this chapter, we have introduced an artifact-based governance policy support for M2M infrastructures. Such a governance artifact is composed of seven modules dedicated to specific governance tasks.

- The *Governance Monitoring* module observe the M2M infrastructure to control its functioning and the requirements fulfillment. *Governance Properties* provide means for the governance tactic agents to monitor the governance performance, while *Governance Events* alert the agents of critical failures to handle. Then, *Governance Operation* allows the agents to trigger governance actions to be performed by the artifact and to set up governance policies. Such governance actions operate on the M2M infrastructure by the *Governance Controller* module. They can be coordinated and/or propagated to other parts of the M2M infrastructure through *Governance Links*. Finally, the *Governance Policy Engine* enables the artifact to run a control loop and make simple local governance decisions.

- Such local governance control loop is specified by the means of *Governance policies*. They are defined by a set of ECA rules. Such ECA rules follow a language defined in order (i) to react to *M2M policy events*, (ii) given a specific *M2M policy context*, and (iii) trigger *M2M governance actions*.

Using such governance policy artifacts, the agents can evaluate the governance and its performance regarding the strategy objectives. When it does not fulfill the objectives, they can decide to adapt the governance policy. They can either enforce and relax governance parameters or set up new governance policies to handle the problems differently.

Analyzing the results of such governance policy and the M2M infrastructure events, the agents can also adapt the governance tactic and strategy. Hence, the governance can evolve with respect to the M2M infrastructure evolution.

This chapter concludes our MAOP approach for an M2M governance. In the next part, we evaluate such a proposal. Chapter 9 presents a JaCaMo based implementation of an M2M governance for the SensCity project. Then, we provide simulation-based experimental results to sustain our approach in Chapter 10. Finally, Referenceschap:conclusion synthesizes our contribution and discuss it to draw further work tracks.

Part III

EVALUATION

CHAPTER 9

AGAMEMON: a JaCaMo based Implementation

Contents

After the description of our M2M governance proposal, we present in this chapter, a proof-of-concept implementation of the AGAMEMON governance based on the JaCaMo framework. This implementation is used to govern the SensCity project M2M infrastructure.

JaCaMo is a MAOP framework based on three frameworks dedicated to separate dimensions of the MAS: (i) *Jason* to program BDI agents, (ii) CArtAgO to program environment artifacts, and (iii) $\mathcal{M}$OISE to specify and run the organization. Each of these frameworks are integrated together so that they can interact with each other.

Thus, JaCaMo is a convenient framework for developing MAS along the vowel dimensions. And so, it seems to be an appropriate technology to support our M2M governance model.

SensCity is a collaborative project lead by Orange Labs to experiment a city-scale deployment of an M2M infrastructure in Grenoble, between 2009 and 2012. Such an experiment was the base scenario and motivation for this thesis. Hence, while the wireless technology used (i.e. Wavenis) and the experiment results were promising, it also faced some issues.

For example, one of the first objective was to provide an horizontal M2M infrastructure, stakeholders were reluctant to share their devices' access by fear they would loose control over it. Finally, only the network resources –WSAN repeaters and the M2M gateway– and the core platform were shared, while the other stakeholders provided both the devices and the associated services.

As a consequence, we decided to integrate the AGAMEMON governance framework to the SensCity platform to demonstrate the feasibility of such an horizontal governance.

In this chapter, we describe how we implement the M2M governance as a MAS and how it is integrated to a real M2M platform. First, we detail the AGAMEMON governance implementation in Section 9.2, after having described the JaCaMo framework in Section 9.1. Then, we provide a brief description of the SensCity architecture and how we define its governance and deploy it, in Section 9.3.1. Finally, in Section 9.4, we synthesize the work presented in this chapter.

9.1 JaCaMo framework

As mentioned before, we decide to use the JaCaMo framework to implement our governance platform as it provide the possibility to program a MAS along the vowel dimensions of the MAS (see 3.1) we used for each of the governance layers we proposed in Part II.

As a matter of fact, JaCaMo is a synergistic combination of three frameworks: (i) $\mathcal{M}$OISE (organization), (ii) *Jason* (agent), and (iii) CArtAgO (environment) frameworks, as represented by Figure 9.1.

$\mathcal{M}$OISE is a organization-oriented programming platform providing support for (i) structured organization based on roles, missions and norms, and (ii) agent-organization interactions. *Jason* is an agent-oriented programming platform supporting (i) the BDI agent reasoning cycle, and (ii) agent-agent interactions. CArtAgO is environment-oriented programming platform supporting (i) environment workspace to run and distribute

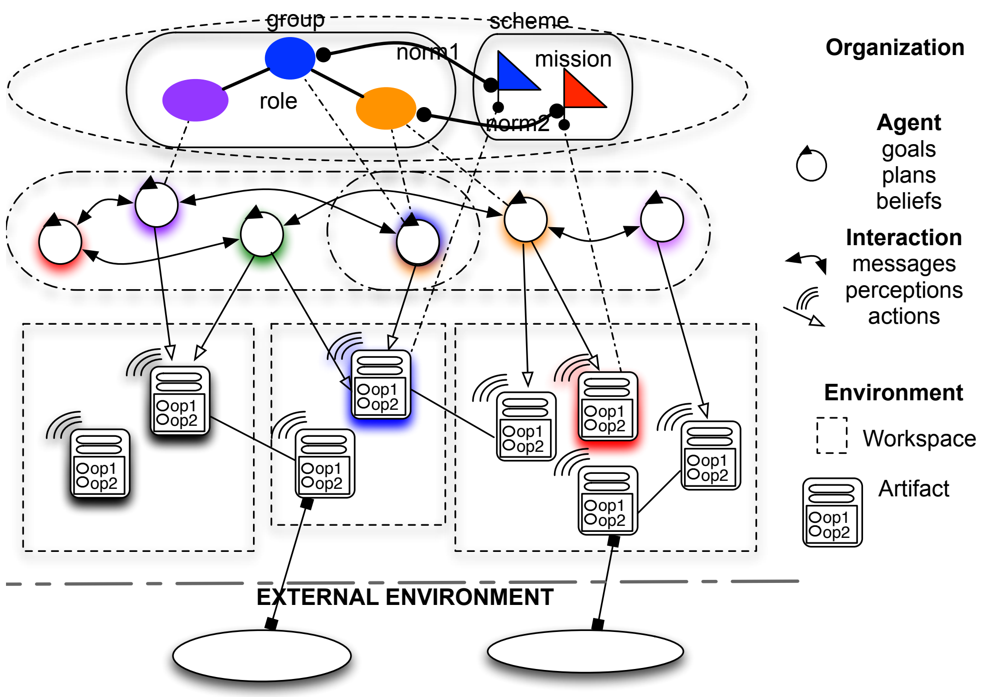


Figure 9.1: JaCaMo abstraction layers: the *organization* is managed by the MOISE framework, the *agents* are programmed in *Jason* and *environment artifacts* are developed with CArtAgO abstracting the *external environment resources*.

the MAS, (ii) artifacts as an abstraction of environment entities, including external systems, and (iii) agent-environment and environment-environment interactions.

Each framework is used to program a dimension of the MAS. Then, concept mapping is performed by the interaction dimension to bind each dimensions together. The synergy between each framework allows developers to program all parts of the MAS separately and combine them with a minimal effort.

In this section, we review how we can develop a MAS in each dimensions using the JaCaMo framework: the organization with MOISE in Section 9.1.3, the agents with *Jason* in Section 9.1.1 and the environment with CArtAgO in Section 9.1.2.

9.1.1 *Jason* framework

Jason is an extension of the AgentSpeak(L) language interpreter implemented in Java. That is, it allows to define an agent's behavior by a program which is interpreted and run by the *Jason* engine. Here, we focus *Jason* elements used to define the agent program. Interested readers may refer to Bordini et al. (2008) for further details about the *Jason* architecture and execution cycle.

In this section, we first describe briefly the AgentSpeak(L) architecture, before focusing on how we can program agent in *Jason* and, finally, extend the *Jason* language to our specific needs.

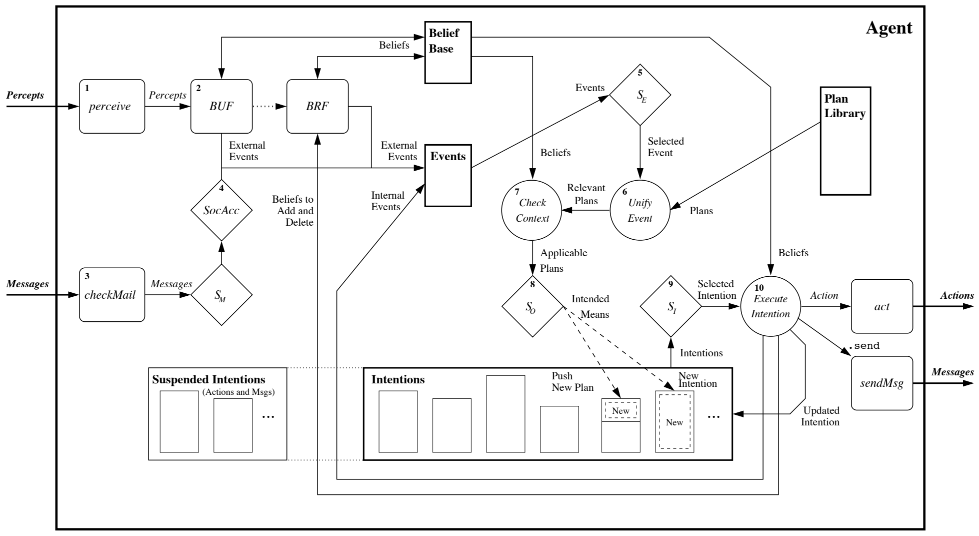


Figure 9.2: AgentSpeak(L) BDI execution cycle.

AgentSpeak(L) BDI Cycle. AgentSpeak(L) is a logic-based agent programming language based on the Belief–Desire–Intention (BDI) architecture proposed by Rao (1996). We do not intend to provide an extend description of the AgentSpeak(L) language, as it is out of the scope of this chapter, but we identify the main component and steps of the agent reasoning cycle.

Figure 9.2 represent the execution cycle of such an architecture, rectangles represent data structures that contain the main components that determine the agent state (i.e. belief base, set of events, plans library and set of intentions), while rounder boxes, diamonds and circles represent functions used in the reasoning cycle. Without being extensive, we can outline such a cycle as follows:

1. the agent is notified of *events*, either (i) internally from the execution cycle, (ii) locally by its own *perception*, or (iii) from other agents' *messages*,
2. which trigger *plans* selected according to the relevance given by the agent's *desires* and *context* information contained in its beliefs base,
3. then, such plans are instantiated as *intentions* which are executed concurrently,
4. an intention *execution* may trigger (i) *beliefs updates*, (ii) other *intentions to be triggered*, (iii) external *action to perform*, and (iv) *messages to send* to other agents.

***Jason* Agent Oriented Programming Language.** Thus, AgentSpeak(L) provide means to program high level reasoning software agent, which make it very accurate for our governance purpose. We now describe the *Jason* framework used to implement our agents. As for governance tactics implementation concerns, *Jason* provides the following elements to program the agent's behavior[1]:

[1]For readability purpose, we simplify the description and avoid some details to focus on the main *Jason* programming components. Interested readers might refer to Bordini et al. (2008) for further information.

- **Beliefs** are information the agent has about itself and the MAS around it –e.g. environment state, other agents' intention, organizational facts. Being a *belief* instead of simple internal property, such as in Object Oriented Programming, makes it explicit the state is only supposed to hold such a value but might be out of date or inaccurate.

  In *Jason*, beliefs are represented in symbolic form as predicate representing a *property* –$property(object)$– or a *relationship* –$relation(object, object)$– believed to be hold regarding one or several objects.

  In *Jason*, a belief can be **annotated** by complex terms providing details that are strongly associated with the belief. Annotations are enclosed in square brackets immediately following a literal: $property(object)[annotation]$.

- **Goals** express the properties of the states of the world that the agent *wishes to bring about*. That is, a goal represents a belief the agent *desires* to be true. In *Jason*, there are two types of goals: (i) achievement goals and (ii) test goals.

  An *achievement* goal is denoted by the ! operator. For example, $!own(house)$ means the agent has the goal to achieve the state of affairs it believes it owns a house ($own(house)$).

  A *test* goal is denoted by the ? operator. It is usually use to retrieve an information that will be unified with a literal. For example, the test goal $?value(V)$ will unify the variable V with some value, either contained in the belief base or inferred from other beliefs.

- **Plans** are sequences of actions to perform to react to some events depending on the context. When a plan is selected, it is added to the agent's *intentions*, that is, it becomes active and can be selected by the intention selector (diamond **9** in Figure 9.2). In *Jason*, a plan is denoted by the following syntax:

  $$triggering\_event : context < -body. \tag{9.1}$$

  The *trigerring_event* can be either a change in the beliefs –e.g. a percept– or a goal that has been activated. While such a syntax is rather reactive, this allows to define both (i) the *reactive* agent's behavior –i.e. reacting to a change of beliefs– and (ii) the *proactive* agent's behavior –i.e. reacting to a change of goals. In addition, such changes can be of two types: (i) *addition*, which is denoted by the $+$ operator, and (ii) *deletion*, denoted by the $-$ operator. This leads to six types of triggering events, listed in Table 9.1, where l is a literal corresponding to the event.

  The *context* of a plan is used for checking the current situation so as to determine whether a particular plan, among the various alternative ones, is likely to succeed in handling the event, given the latest information the agent has.

  Finally, the *body* of a plan corresponds to the agent's behavior when the plan is executed. In *Jason*, the body is a sequence of actions to be performed. Such an action can be (i) a simple *action* performed by the agent's effectors, (ii) an *internal action* performed internally by the agent, (iii) an *achievement goal* adding a new goal which can trigger a new plan (also called *sub-goal*), (iv) a *test goal* generally used to unify a belief with variable, or (v) a *mental note* corresponding to a belief addition, annotated with the source $self$.

Notation	*Name*
$+l$	Belief addition
$-l$	Belief deletion
$+!l$	Achievement-goal addition
$-!l$	Achievement-goal deletion
$+?l$	Test-goal addition
$-?l$	Test-goal deletion

Table 9.1: Types of triggering events in *Jason* plans.

***Jason* Developer Library.** Due to such a reasoning cycle, the *Jason* framework provides an interesting means for programming agents capable of governance tactic decisions. Further more, *Jason* provide a developer library which can be used to customize many parts of the agents' cycle. This makes *Jason* very flexible and suitable for integration with different platforms, such as CArtAgO and *M*OISE, for example. We can use such a developer library to provide additional capacities to the agents in order to be used for governing the M2M infrastructure.

The *Jason* library allows developers to program custom components of the *Jason* interpreter, such as agent architectures, reasoning cycles, agents' belief base or pre/post-processing functionalities. In particular, we can define custom *internal actions*, which we focus on in this section.

An internal action is an action which is run internally within an agent, and so, without any change on the environment. This is used to extend the programming language with operations that are not otherwise available and to provide an easy and elegant access to legacy code.

In *Jason*, an internal action is called by the name of the library, followed by a `'.'` and the name of the internal action, as follows:

$$math.manathanDistance(X1, Y1, X2, Y2, D) \tag{9.2}$$

Such an internal action can be programmed by implementing the `InternalAction` interface. This interface is composed of two methods: (i) *`suspendIntention():boolean`* and (ii) `execute(TransitionSystem,Unifier,Term[]):Object`. The first method allows to set the agents' intention to be suspended –returning `true`– while performing the internal action. The second method is called by the AgentSpeak(L) interpreter to execute the internal action. Its first parameter contains the current state of the agent. The second one is the unifying function to be used to use values bound to AgentSpeak(L) variables. Finally, the third method is an array of terms containing the internal action's parameters.

The *Jason* library provides a class, called `DefaultInternalAction`, that simplifies the programming of custom internal actions. Finally, let us note that the library –i.e. Java package– and the internal action –i.e. Java class– names must starts by a lower case letter, as identifiers starting with an uppercase letter are variables in AgentSpeak(L). Listing 9.1 provides an example of an internal action implementation, corresponding to the example given above.

```java
package math;                                          // Internal Action Library Name
public class manathanDistance extends DefaultInternalAction {
  public Object execute(TransitionSystem ts, Unifier un, Term[] args)
                                                              throws Exception {
   try {
    NumberTerm p1x = (NumberTerm)args[0];     // 1. gets the arguments as typed terms
    NumberTerm p1y = (NumberTerm)args[1];
    NumberTerm p2x = (NumberTerm)args[2];
    NumberTerm p2y = (NumberTerm)args[3];
    double r = Math.abs(p1x.solve()-p2x.solve()) +      // 2. calculates the distance
               Math.abs(p1y.solve()-p2y.solve());
    NumberTerm result = new NumberTermImpl(r);          // 3. creates the result term
    return un.unifies(result,args[4]) // 4. unifies the result with the 5th argument
   } catch (ArrayIndexOutOfBoundsException e) {
     throw new JasonException("Five arguments needed");
   } catch (ClassCastException e) {
     throw new JasonException("Some arguments that are not numbers");
   } catch (Exception e) {
     throw new JasonException("Error in 'distance'");
}}}
```

Listing 9.1: Example of an Internal Action implementation: `math.manathanDistance`.

9.1.2 CArtAgO framework

CArtAgO is a Java implementation of the Agents & Artifacts (A&A) meta-model. As described in Figure 9.3, it provides components to program the environment and to enable agents to interact with it. The environment is represented by one or several workspaces, possibly distributed among different nodes. Each workspace is composed of artifacts which provide a first-class abstraction of environment entities to be used by the agents.

In this section, we first overview the CArtAgO framework's main components and, then, we describe how artifacts can be used by agents.

CArtAgO Artifact. Artifacts are resources and tools dynamically constructed and manipulated by agents to support their individual and collective activities. Hence, an artifact is used to structure and organise the environment. It provides a general-purpose programming and computational model to embed functionalities available to agents.

CArtAgO provides tools for programming artifacts which are composed of: (i) a set of *observable properties*, (ii) a set of *observable events*, and (iii) a set of *operations* as described in Figure 9.3 –we do not consider the *manual* as it is not relevant to our purpose.

An *observable property* represents a state variable that can be perceived by the agents observing the artifact. The state variable's value is bound to the agent's belief base through its perception module. Such a value can be changed dynamically during operations' processes.

An *observable event* represents non-persistent events occurred inside the artifact. As observable properties, such events are triggered during the operations' processes and are perceived by the agents.

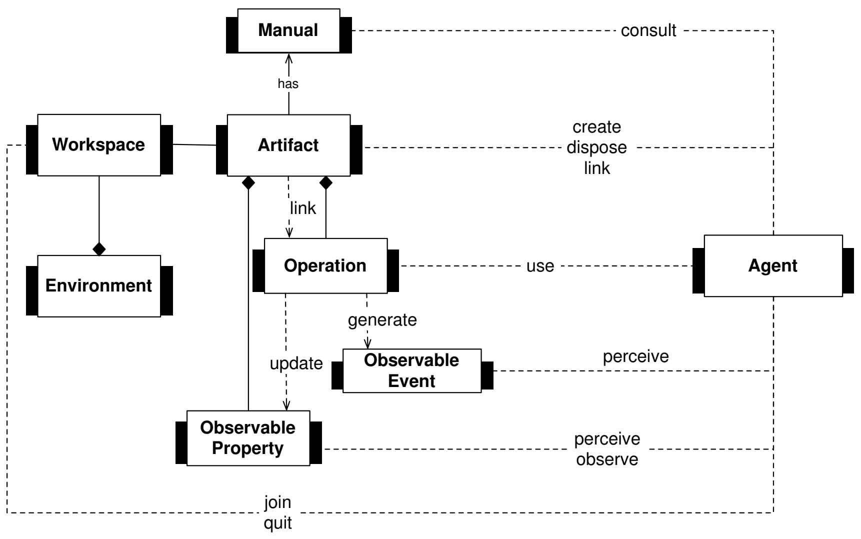


Figure 9.3: UML-like representation of A&A main concepts in CArtAgO.

An *operation* is a computational processes executed inside artifacts. It can be triggered by agents or other artifacts, when linked to them.

CArtAgO Workspace. A workspace represents a place an agent is situated in –an agent can possibly be situated in several workspaces at the same time. It contains a set of artifacts the agents can work with. As a matter of fact, the workspace itself is represented by a *WorkspaceArtifact* artifact, which provides the main functionalities to work within the workspace –i.e. `create`, `dispose`, `lookup`, `link`, `unlink` and `focus` artifacts.

A workspace can be spread among several network nodes, which can host several workspaces. By default each node has a default workspace where agents are situated at bootstrap. Nodes are represented by *NodeArtifact* artifacts, which provide means to `create` a new workspace, to `join` and to `leave` a local or a remote workspace.

Artifact usage by Agents. As mentioned before, CArtAgO artifacts provide an environment abstraction to the agents. Thanks to such an abstraction, CArtAgO allows to develop open and heterogeneous MAS using different technologies and interacting via the artifacts. As matter of fact, CArtAgO is integrated with many agent programming platforms, in particular Foundation for Intelligent Physical Agents (FIPA) (FIPA, 2000) compliant ones.

Regarding our development purpose, CArtAgO offers two interesting integration means. On the one hand, the CArtAgO library comes with a `cartago.util.agent.Agent` class which enables to easily manipulate artifacts in pure Java programs. On the other hand, a *Jason* wrapper allows to program AgentSpeak(L) agents which percepts and external actions are bound to artifact properties, events and operations.

Using such interfaces, we can use artifact to enable governance agents to monitor and control the M2M infrastructure. This will be detailed further in Section 9.2 and Section 9.3.

```java
public class MessageQueueArt extends cartago.Artifact {
    Queue<String> queue;
    @OPERATION void init(int capacity) {
      this.queue = new java.util.concurrent.ArrayBlockingQueue<String>(capacity);
      defineObsProperty("capacity",capacity);
      defineObsProperty("size",0);
   }
   @LINK void put(String message){
      if(this.queue.remainingCapacity() > 0) {
         this.queue.add(message);
         getObsProperty("size").updateValue(prop.intValue()+1);
      } else
         signals("Queue is full , message lost");
   }
   @OPERATION get(OpFeedbackParam message) {
      String m = this.queue.poll();
      if(m == null) {
         signals("No message in queue.");
         return;
      }
      message.set(m);
      getObsProperty("size").updateValue(prop.intValue()-1);
} }
```

Listing 9.2: Example of an artifact implemented with CArtAgO: *MessageQueueArt*.

9.1.3 Moise framework

Last part of the JaCaMo framework, Moise is an organization-oriented framework to program organizational structures that constraints the agents to achieve a global objective. An organization programmed in Moise is defined by (i) a *structural specification*, (ii) a *functional specification* and (iii) a *normative specification* that binds the two previous specifications.

In this section, we first describe the OS along these three dimensions, and then, we describe how the OE is supported at run-time by the JaCaMo framework.

Structural Specification. The structure of an organization in Moise is defined along the concepts of *roles*, *relationships* and *groups*.

A *role* refers to the status in the organization. On the one hand, such a status assigns individual responsibilities to the agents which have adopted it. On the other hand, it also assigns collective responsibilities to the agents, through the role's relationships with the other roles and the deontic specification to the functional specification.

A *group* is an intermediate structure responsible for collective tasks –i.e. *social schemes*, described below. A group is composed of non-abstract roles and sub-groups. For each of these roles and sub-groups, a *cardinality* is defined to consider the group well formed. The group also defines *compatibility* constraints between roles either within the same group –i.e. *intra-group* compatibility– or between different groups –i.e. *inter-group* compatibility.

A *relationship* between two roles is defined by three types of oriented links. The *acquaintance* link provides representation of the agent playing the target role to the agent playing the source role. The *communication* link allows the agent playing the source role to communicate with the agents playing the target role. Finally, the *authority* link makes the agent playing the source role able to delegate goals, plans and actions to the agent playing the target role, and obliges the latter to commit to them.

Such relationships can be specified at the *intra-group* level and/or the *inter-group* level.

Functional Specification. The functional specification in Moise defines the collective behavior of the organization. It is structured into *social schemes*. A social scheme is composed of (i) a plan and (ii) a set of missions.

A *goal* describes a task to perform or a state to reach. In Moise, two types of goals are defined: (i) *achievement goals*, which are considered satisfied once the agents have fulfilled it, and (ii) *maintenance goals*, that agents should constantly maintain satisfied.

A *plan* defines the means to fulfill a *root goal* through a decomposition of sub-goals. Such a decomposition can be (i) a *sequence* of goals to fulfill successively, (ii) a *choice* between several goals –i.e. different alternatives– or (iii) a *parallelism* which consists of several goals to fulfill concurrently. In addition such a tree decomposition, the plan defines, for each goal, a *cardinality* of agents that should fulfill it to be satisfied by the organization.

A *mission* is a consistent set of goals to fulfill. It groups goals that should be treated together, and so, be assigned to the agents at once. In fact, in Moise, goals are assigned to the agents through deontic relations between roles and missions defined by the normative specification.

Normative Specification. In Moise, *norms* are used to bind the structural and functional parts of the organization. A norm defines a *deontic relation* –i.e. an *obligation* or a *permission*– between a *role* and a *mission* which is triggered when a specific *context* is hold. Such a norm forces the agents playing the corresponding role to commit to a mission within a specific time –also called *time to fulfill*.

Such norms allows to define the structure and the task of the organization independently and to bind them dynamically at run-time.

Organizational Entity. The OE represents an instance of an OS at run-time. That is, an OE is composed of (i) a set of agents committed to the roles, (ii) a set of schemes instantiated by the different groups of agents, and (iii) a set of active norms regulating the agents' activity. Such an OE is integrated into the JaCaMo framework by (i) an organization-environment mapping and (ii) an organization-agent mapping.

Organization-Environment mapping. In such an approach, the different concrete computational entities –managing the current state of the organisation in terms of groups, social schemes, and normative state– are reified, outside the agents, by means of *organisational artefacts*, namely the *GroupBoard* and the *SchemeBoard* artifacts. Such artifacts encapsulate and enact the organisation behaviour as described by the OS.

From an agent point of view, such organisational artifacts provide those actions that can be used to proactively take part in an organisation –e.g. adopt/leave particular roles, commit to missions, mark a social goal as achieved, etc. They also provide specific observable properties dynamically to make the state of an organisation perceivable along with its evolution. Besides, they provide actions that can be used by organisational agents to manage the organisation itself. For example, a *ReorgBoard* artifact allows the agents to dynamically start/interrupt the OE and change the OS.

Organization-Agent mapping. In JaCaMo, the organization and agent dimensions are bound together by the means of *goals*. Goals defined at the organisation dimension –i.e. goals that are to be achieved by the organisation– are mapped into individual agent goals, which agents may decide to adopt or not. This delegation of goals from the organisation to the agents is expressed by obligations. The state of goals from the organisation dimension (which agents should fulfil them and when) is maintained by the organisational artifacts and is displayed as obligations for the agents. An obligation is fulfilled when the corresponding goal is achieved by the agent before the deadline.
At the agent side, when such an obligation is perceived, and the agent chooses to adopt it, a corresponding (individual) agent goal is created. It should be noted that, besides being free to adopt or not the goals from the organisation, the agent is also free to decide which courses of action should be used to achieve each goal, and that in turn might mean adopting further individual goals.
As we can see, the mapping of goals prescribed by the organisation into agent individual goals is under the complete control of the agent's decision making process (so as to preserve its autonomy).

9.2 A JaCaMo Implementation of AGAMEMON

In the previous section, we introduced the JaCaMo framework and showed how it suits to the development of a MAS. We now describe how we implement the AGAMEMON governance framework using the JaCaMo framework.

We first describe the governance tactics implementation in *Jason*, in Section 9.2.1. Then, we present the horizontal reorganization implementation in Section 9.2.2 and the vertical SLA reasoning process in Section 9.2.3. We end this section describing how we embed governance policies in CArtAgO artifacts, in Section 9.2.4, before describing the AGAMEMON governance of the SensCity infrastructure in the next section.

9.2.1 Governance Tactics support in *Jason*

As stated in Chapter 7, we choose to implement governance tactics as agent plans. We have defined three types of agents which address different M2M entities specific governance concerns, namely the application, the Telco infrastructure and the M2M devices –respectively *AppAg*, *InfAg* and *DevAg* agents. We have defined a base governance agent architecture –the *GovAg* agent– which implements the common governance reasoning cycle.

Such a governance cycle can be summarized as follows: (i) each type of agent adopts different roles in horizontal and vertical organizations, binding the horizontal and vertical requirements, corresponding to the strategic objectives they take the responsibility of; (ii) then, agents monitor the M2M infrastructure via governance artifacts and set up regulation policies when problems are detected (iii) reorganization is triggered when either regulation policies fail to make the M2M infrastructure fulfill the strategy objectives (vertical reorganization) or when the number of agents should evolve to govern the M2M infrastructure (horizontal reorganization).

Hence, each governance tactic defined in Chapter 7 is implemented by sets of *Jason* plans. In this section, we focus on how governance agents bind and validate the strategy requirements. We provide an example of code in Listing 9.3. Such a listing shows how governance agents commit to the horizontal and vertical organizations, as implemented by the generic *govAg* agent.

When a governance agent is started, its initial goal `!commitToHorizontalStrategy` (Line 45) makes the agent committing to the horizontal organization. For each of the horizontal requirements the agent believes it should handle, it commits to the corresponding roles. Such a role `R` is defined by the belief `my_role(h,R)` in the specialized agent's code (Line 82) –`h` stands for *horizontal*, while `v` for *vertical*.

Vertical commitment plans are triggered when the agent receives a `recruit` message (Line 51), provided that it addresses the M2M entity the agent governs. Proposal response plans are `atomic`, meanings the agent will suspend all other intentions until plans are totally executed. The agent evaluates the feasibility of the proposed SLA (Line 59) to determine its response –presented in Section 9.2.3. If the SLA is refused, it directly sends the corresponding response (Line 62).

If the SLA is accepted, the agent will first evaluate its load capacity in order to determine if it can handle the governance. This is performed by the `.check_load_availability` custom internal action which takes a `Task` as an input and returns the actual agent's load –based on its number of intentions, i.e. active plans– and the feasibility of the task as output parameters (Line 67).

If the agent is unable to handle the governance load, it triggers an horizontal reorganization, which we detail in the next section.

9.2.2 Horizontal Reorganization in *Jason*

In order to fit the size of the governance layer to the M2M infrastructure's dynamics, we reorganize the horizontal OE by the means of coalitions. A coalition is an informal organization grouping agents of the same type which distribute governance tasks among each other, as described in Section 7.1.3.

When starting, the MAS is composed of one agent of each type, called the bootstrap agent. Such a bootstrap agent detects the registration of M2M entities and creates agents to govern each of them. When an M2M entity requires too much governance activity, the agent governing it creates additional agents. Such an example is given in Listing 9.3, Line 76, where the agent's load is too high to govern an additional vertical contract.

To evaluate the governance workload, we define an heuristic based on the number of intentions, that is, the number of goals to fulfill in order to perform a governance task. In addition, we limit the agents' workload, by setting an arbitrary bound of concurrent intentions agents should avoid to overpass.

Hence, the `.check_load_availability` custom internal action (Listing 9.3, Line 67) (i) counts the agent's number of intentions, corresponding to its actual workload `L`, and (ii) estimates the number of intentions `LT` to fulfill the task `T` given in parameter, then it returns (iii) the agent's actual workload `L` and (iv) a boolean value corresponding to the agent's capacity to handle the task `T`.

```
%% File: govAg.asl %%
!commitToHorizontalStrategy.
+!commitToHorizontalStrategy
 : my_role(h,Role)
<- lookupArtifact("horizontalOrgBoard",HOrgId);
   adoptRole(Role)[artifact_id(HOrgId)].

+recruit(GpID,OSFile,Role,M2M_ENTITY_ID)[source(S)]
   : my_entity(M2M_ENTITY_ID)
  <- !respondVProposal(S,GpID,OSFile,Role,M2M_ENTITY_ID).
+recruit(GpID,OSFile,Role,M2M_ENTITY_ID)[source(S)]
  <- -recruit(GpID,OSFile,Role,M2M_ENTITY_ID)[source(S)].

@atomic
+!respondVProposal(PS,GpID,OSFile,Role,M2M_ENTITY_ID)
<- ?evalSLA(OSFile,R);
   !sendResponseVProposal(PS,GpID,OSFile,Role,M2M_ENTITY_ID,R).
@atomic
+!sendResponseVProposal(PS,GpID,OSFile,Role,M2M_ENTITY_ID,refuse)
<- .send(PS,tell,recruit_response(GpID,OSFile,Role,M2M_ENTITY,refuse)).
@atomic
+!sendResponseVProposal(PS,GpID,OSFile,Role,M2M_ENTITY_ID,R)
                         // R = accept | tradeoff(SLAProposal)
 : .check_load_availability(play_role(Role,GpID),Load,true)
<- .commit_delay(1000,play_role(Role,GpID), Tdelay);
   .wait(Tdelay);
   !tryRole(Role,GpID).
@atomic
+!sendResponseVProposal(PS,GpID,OSFile,Role,M2M_ENTITY_ID,R)
 : .check_load_availability(play_role(Role,GpID),Load,false)
<- .commit_delay(1000,play_role(Role,GpID), Tdelay);
   .wait(Tdelay);
   !hreorg(handle_Vproposal(PS,GpID,OSFile,Role,M2M_ENTITY_ID,R)).

+handle_Vproposal(PS,GpID,OSFile,Role,M2M_ENTITY_ID,R)[source(S)]
<- adoptRole(Role)[artifact_id(GpID)];
   .send(PS,tell,recruit_response(GpID,OSFile,Role,M2M_ENTITY,R)).

    %% File: devAg.asl %%        %% File: infAg.asl %%       %% File: appAg.asl %%
    my_role(h,rRAR).             my_role(h,rGC).             my_role(h,rAE).
    my_role(h,rREM).             my_role(h,rTOE).            my_role(h,rSec).
    my_role(h,rCS).              my_role(h,rIP).             my_role(h,rCB).
    my_role(h,rTM).              my_role(h,rHDR).            my_role(v,application).
    my_role(h,rGC).              my_role(h,rSec).
    my_role(h,rCB).              my_role(h,rCB).
    my_role(h,rSec).             my_role(v,corePlfm).
    my_role(v,sensor).           my_role(v,gateway).
    my_role(v,actuator).
    my_role(v,repeater).
```

Listing 9.3: Common Role Commitment plans in *GovAg* agents and specialization by *AppAg*, *InfAg* and *DevAg* agents.

Based on such a workload estimation, agents whether to handle the governance task or to scale-out the agent layer. This is performed through an horizontal OE reorganization, consisting in the creation of a new agent to govern the same M2M entity.

On the one hand, this agent adopts the same roles in the horizontal reorganization. On the other hand, it directly handles the vertical task which could not be handled previously. As expressed in Section 7.1.3, such a vertical task could consist in (i) governing a vertical contract –i.e. adopting a role in the vertical organization–, (ii) governing a sub-set of requirements in this contract –i.e. committing to particular missions–, or (iii) performing governance tasks for a specific requirement –i.e. realizing sub-goals to fulfill an organizational one.

Several agents share the governance of the same M2M entity correspond to a *governance coalition*. As it is an informal organization, we define coordination mechanisms so that they can distribute the governance tasks among each other. To do so, we use a greedy heuristic, implemented by the `.commit_delay` internal action (Listing 9.3, Line 68). This internal action computes the delay before committing, defined in Equation (7.1) p. 138, as follows:

$$T_{delay} := TC - TC \times \frac{ML - EW}{ML}$$

Due to such an heuristic, the least loaded agent waits less time before handling a new governance task, and other agents do not commit to it. In our example, the intention `!tryRole(Role,GpID)` is triggered after waiting a `Tdelay` delay (Line 70). The corresponding plans check whether an other agent already committed to the role or not. If not, the agent adopts the role in the vertical organization, using the *OrgBoard* artifact operation `adoptRole(Role)`. The same verification is performed before creating a new agent –`!hreorg` (Line 76).

9.2.3 SLA QoS Attributes Analysis

Vertical SLAs are turned into an OS following a template defined in Section 6.3. Such a template is a declarative specification of vertical strategy objectives allowing the agents to be aware of them.

Nevertheless, $\mathcal{M}$oise does not permit the agents to access the whole OS before the OE schemes are instantiated. Yet, during the SLA validation process (see Section 6.4)) the agents should validate such an SLA before instantiating the organization. In fact, OE instantiation corresponds to a "virtual contract agreement".

Hence, we provide governance agents the capability to read and evaluate such an SLA from a $\mathcal{M}$oise OS file, as depicted in Listing 9.4. To do so, we provide governance agents an internal action to read an SLA OS file and to parse M2M requirements and QoS parameter. Listing 9.5 presents such an `slaFromOS` internal action.

For clarity purpose, we do not detail the `parseSLAfromOS()` method in this manuscript, as it is just an XML parser method. This method simply extracts goals' description field and put them in a Java map containing the goals' name –i.e. requirements– as keys and descriptions –i.e. QoS parameters– as values.

The main code of this internal action turns each $\mathcal{M}$oise goals into a literal describing the requirement and its QoS parameter. Then, it unifies the output parameter with such a literal.

```
<goal id="environmentSensing" ds="sensing_rate(1/hour)" />
```

Listing 9.4: Example of requirement definition in an SLA expressed as an XML *M*OISE OS, corresponding to an *environment sensing* requirement, with a QoS value of *one message per hour rate*.

```
package parseQoS;
/** parseQoS.slaFromOS Internal Action
 * Unifies a QoS description string with the corresponding Literal
 * Syntax: parseQoS.slaFromOS(osFile, slaLiteral)
 * Input:  osFile:: MOISE XML Organizational Specification
 * Output: slaLiteral:: sla([rq([qosParam(qosValue)]*)]*)
 */
public class slaFromOS extends DefaultInternalAction {
  public Object execute(TransitionSystem ts, Unifier un, Term[] args)
                                                      throws Exception {
    Map<String,String> sla = parseSLAfromOS((StringTerm)args[0]).getString());
    StringBuffer slaLit = new StringBuffer();
    slaLit.append("sla(");
    for(String rq: sla.keySet())
      slaLit.append(rq).append("(").append(sla.get(rq)).append(")");
    slaLit.append(")");
    un.unifies(ASSyntax.parseTerm(slaLit.toString()), args[1]);
    return true;
  }
  protected Map<String,String> parseSLAfromOS(String xmlOSFileName) { /* ... */ }
}
```

Listing 9.5: `parseQoS.slaFromOS` Internal Action in Java.

```
+?evalSLA(M2MEntityID, SLA_OS, true)
 <- parseQoS.slaFromOS(SLA_OS, sla(SLA))
    for ( .member(RQ,SLA)) {
      ?eval_rq(M2MEntityID,RQ,true);
    }
+?eval(M2MEntityID, SLA_OS, false).
+?eval_rq(M2MEntityID,RQ(RQV),Valid)
   : concerned(M2MEntityID,RQ)
 <- ?governance_artifact(M2MEntityID, DevArtID);
    isFeasible(RQ(RQV),Valid)[DevArtID].
+?eval(M2MEntityID,RQ(RQV),true)
   : not(concerned(M2MEntityID,RQ)).
```

Listing 9.6: *evalSLA* plan to evaluate feasibility of an SLA in Jason.

```
+m2m(critical_devices_detected(Devices))
<- alert(critical_devices_detected(Devices)).

+m2m(critical_devices_detected(Devices))
:  .length(Devices,L) & param(pDevGpConsistency,C) & L > C
  & not(param(link(degraded_group,unset)))
<- perform(createDegradedDeviceGroup);
   alert(degradedDevicesGroupCreated) ;
   do(m2m(criticalDevicesDetected)).

+m2m(critical_devices_detected(Devices))
:  param(link(degraded_group,L)) & ~param(link(degraded_group,unset))
<- perform(migrate(Devices,L));
   alert(devicesMigrated(AllDevs,L)).

+policy(changedDeviceNb(N))
:  param(pDevGpConsistency,C) & N < C
  & param(link(degraded_group,L)) & ~param(link(degraded_group,unset))
<- ?policy(device_list(AllDevs));
   perform(migrate(AllDevs,L));
   alert(devicesMigrated(AllDevs,L));
   alert(emptyDeviceGroup).
```

Listing 9.7: *Jason* rules expressing the Device Lifetime Management M2M Governance Policy.

Listing 9.6 describes the agent's plan to evaluate and validate such an SLA. An SLA is validated if and only if each of its QoS parameter is estimated feasible (Line 115–Line 120).

To determine its feasibility, the agents uses the artifact governing the M2M entity to evaluate the cost of such an QoS parameter (Line 124). Each agent only evaluates QoS parameters their governed M2M entity is addressed by. Therefore, it considers all other requirements as feasible (Line 125).

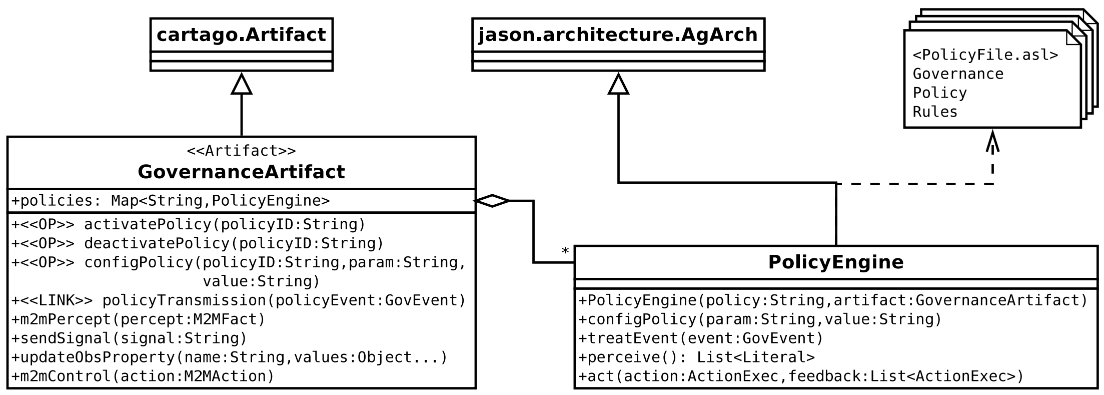
9.2.4 Governance Policies support in CArtAgO

In the three previous sections, we described how we have implemented key parts of the governance agent layer. We now describe how we embed a *governance policy engine* within CArtAgO artifacts, so that they can monitor and control the M2M infrastructure according to governance policies set up.

As stated in Section 8.1.1, we define an M2M governance policy as a set of ECA rules. To do so, we chose to declare such ECA rules as a *Jason* program. Hence, we can use the *Jason* engine to evaluate the rules, while keeping an high degree of abstraction.

As an example, in Listing 9.7 we translated the *Device Lifetime Management* policy presented in Section 8.1.2. It is composed of four rules, each one treating an event within particular conditions.

Let us note that such a *Jason* program is not defined in terms of goals and plans, but each rule only reacts to a perceived event. In addition, rules define only straight actions with no plan decomposition. This matters so that policy ECA rules remain reactive and simple enough to be executed quickly.



cartago.Artifact
jason.architecture.AgArch
<PolicyFile.asl>
Governance
Policy
Rules
<<Artifact>>
GovernanceArtifact
+policies: Map<String,PolicyEngine>
+<<OP>> activatePolicy(policyID:String)
+<<OP>> deactivatePolicy(policyID:String)
+<<OP>> configPolicy(policyID:String,param:String, value:String)
+<<LINK>> policyTransmission(policyEvent:GovEvent)
+m2mPercept(percept:M2MFact)
+sendSignal(signal:String)
+updateObsProperty(name:String,values:Object...)
+m2mControl(action:M2MAction)
*
PolicyEngine
+PolicyEngine(policy:String,artifact:GovernanceArtifact)
+configPolicy(param:String,value:String)
+treatEvent(event:GovEvent)
+perceive(): List<Literal>
+act(action:ActionExec,feedback:List<ActionExec>)

Figure 9.4: UML representation of the Governance Policy Engine implementation in AGAMEMON.

Such governance policies are executed by the *Governance Policy Engine*. In Figure 9.4, we describe our implementation of a such a policy engine to be used by governance artifacts. In this figure, we showed only parts of the artifact which are concerned with the governance policy process.

- Artifact operations –represented by the `<<OP>>` stereotype– allows governance agents to activate, deactivate and configure policies. Each policy is managed by a separate instance of `PolicyEngine`.
- Artifact links –`<<LINK>>` stereotype– allows artifacts to propagate policy facts. These are turned into a policy events –`policy(Event)`– and transmitted to the policy engines.
- Perception modules monitor parts the M2M infrastructure. They translate each `M2MFact` it detects as an M2M event –`m2m(Event)`.

In order to manipulate the *Jason* implemented governance policies, we used the `AgArch` class, from the *Jason* API, as a superclass for the `PolicyEngine` class. When instantiated, the `PolicyEngine` class loads the *Jason* file containing the policy definition and creates the corresponding agent. Each policy is managed by a concurrent instance of `PolicyEngine`, as follows:

1. The *`configPolicy(param,value)`* method enables the governance artifact to parametrize and reconfigure the policy at runtime. Each parameter is represented by a `param(Param)` literal which is added to the belief base.
2. The *`treatEvent(event)`* method is used by the artifact modules so that the policy engine can evaluate them. Such an event is translated as a literal and added to the perception list. Then the method invokes the *Jason* engine to evaluate the policy rules according to the event.
3. The *`perceive()`* method overloads the `AgArch`'s one so that the *Jason* engine uses the governance events previously generated.
4. The *`act(action, feedback)`* method overloads the `AgArch`'s one in order to handle the governance actions defined by the policy. It is invoked by the *Jason* engine when a rule contains an external action to perform.

This method parses the literal corresponding such an action and performs it. An internal action – `do(A)`– generated a new event to be evaluated by the policy. An `alert` action is handle by invoking the artifact to send a signal to the governance agents –*`sendSignal(signal)`* method– which can, potentially, update the artifact's observable properties. M2M actions are operations to be performed on the M2M infrastructure, via the *`m2mControl(action)`* method.

In this section, we have presented the implementation of the main features of our governance framework. We use the JaCaMo framework in order to specify the governance objectives and requirements as $\mathcal{M}$oise Organizational Specification . We have programmed agents in *Jason* to carry out the governance tactics and apply the governance objectives. Finally, we have used CArtAgO artifacts to embed a governance policy engine, which allows the artifacts to automate a control loop over the M2M infrastructure.

9.3 AGaMeMon Governance for the SensCity Project

After having presented our JaCaMo implementation of AGaMeMon, we explain, in this section, how we integrate our governance framework into an existing M2M core platform used for the SensCity project. To do so, we perform the two following steps:

1. We identify the core platform's components that implement and/or contribute to the strategic requirements, the tactic decision and policy actions.
2. We enable the governance artifacts to monitor and control the M2M infrastructure via those components.

First, we briefly describe the SensCity architecture in Section 9.3.1. Then, in Section 9.3.2, we define the SensCity governance. Finally, we describe an hybrid Component-Based Software Engineering (CBSE)–MAOP implementation approach for wrapping the SensCity components with CArtAgO artifacts, in Section 9.3.3.

9.3.1 SensCity Architecture

The SensCity project's infrastructure is designed as an N-tier architecture, as described in Figure 1.2 (see Section 1.2, p. 21). SensCity core platform is deployed as a component-based Web Service server. In this section, we give a brief description of the SensCity architecture. A more detailed description of the core platform components is provided in Appendix B.

Several stakeholders contribute to the different parts of the infrastructure. Such an infrastructure is composed of (i) several M2M client applications providing value-added services to end user customers, (ii) an M2M core platform, divided into two sub-platforms, (iii) a gateway to connect the devices to the platform, (iv) M2M devices providing "business" services (i.e. sensors and actuators), (v) and repeaters to extend the coverage of the WSAN, as described in Figure B.1.

The Telco stakeholder provides the core platform, the gateway and the repeaters for deploying the M2M devices and for enabling M2M services to request and subscribe to the devices.

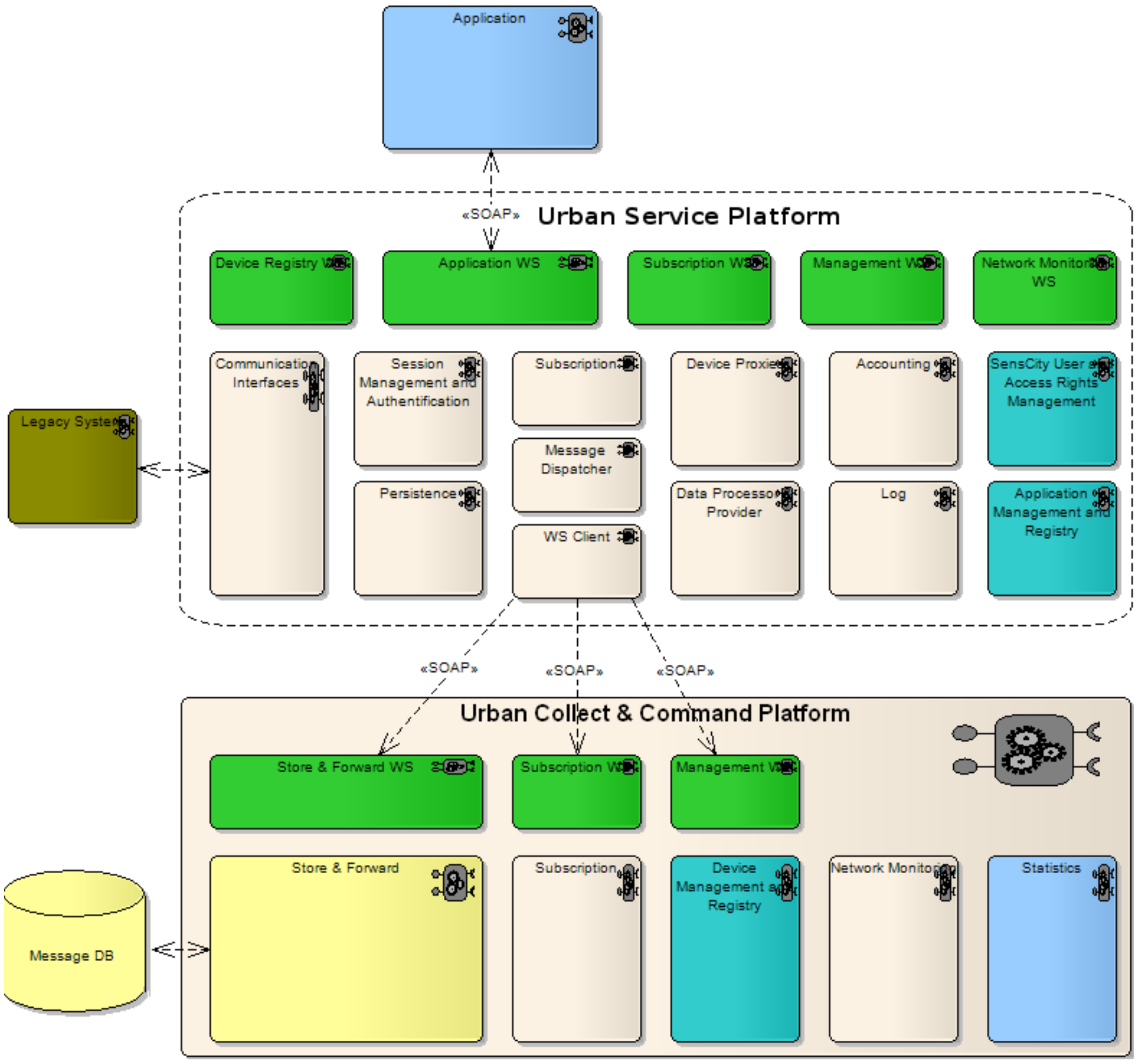


Figure 9.5: The *SensCity's core platform* component-based architecture.

It is divided into an `Urban Service Platform (USP)` part to manage the notifications sent to the applications, their rights and billing, and an `Urban Collect & Command Platform (UCCP)` part to manage the devices and the communication with them. Hence, the core platform is a strategic component of the infrastructure, from a governance point of view. Therefore, it is worth focusing on these two platforms' architecture.

Organization		*Agent*	*Artifact*	*SensCity Resources*
gAEExpose	[H]	—	—	*USP.applicationService*
				USP.configurationService
gAERegister	[H]	*!hReorg(AppID)*	make *AppArt*	*USP.sessionService*
		!registerApp(AppID)	*AppArt.validRegistration*	*USP.sessionService.register(AppID)*
gAERouting	[H]	—	—	*USP.applicationService*
gAERecord	[H]	—	—	*USP.hibernate*
startSch(schRecruit)	[H/V]	*+slaProposal(SuID)*	*AppArt.signal(slaProposal)*	*USP.subscriptionWS*
createOE(vM2M-SuID)	[V]			
gValidSLA	[H/V]	*adoptRole(application)*	—	—
gDeploySLA	[H/V]	*!deployDataCollect*	*AppArt.validSLA(SuID)*	*USP.subscription*
startScheme(schDataCollect)	[H/V]			*USP.applicationConfiguration*
gCBBill	[H]	*!check_app_rights*	*AppArt.monitorAppViolations*	*USP.messageDispatcher*
computePenalties(V)	[V]			
gCBRedeem	[H]			
billCompensation	[V]	*!bill_app*	*AppArt.bill()*	—

Table 9.2: Example of mapping between governance horizontal/vertical strategy (*Organization*), tactics (*Agents*), policies (*Artifact*) and SensCity resources.

Business Components. The external "business" components –i.e. devices and value-added services– are provided and managed by the other stakeholders. The Telco operator has to manage the whole infrastructure in order to guarantee its properties such as QoS and availability, even if it has no direct control over the "business" components.

We call "business" components the parts of the M2M infrastructure dedicated to the "business" functionalities of an M2M service. These are the *applications* and the *devices*.

Urban Service Platform Components. The *Urban Service Platform* (*USP*) manages the application domain of the M2M infrastructure.

Its components take care of the *application* information and activity. That is *application* registration, their subscriptions to M2M devices and M2M data retention.

It provides a web service interface for the M2M applications to register, subscribe to M2M devices, and receive notifications. Another web service interface is used to receive M2M devices information from the *UCCP* platform.

Urban Collect & Command Platform Components. The *Urban Collect & Command Platform* (*UCCP*) manages the device domain if the M2M infrastructure.

Its components manage information about M2M devices, such as WSAN routes and gateways for message transmission. It also contains network monitoring components and maintains statistics about the devices.

It provides web services to M2M gateways to enable message transmission from and to the M2M devices. Another web service provides means to the *USP* for enabling subscriptions M2M devices.

9.3.2 SensCity Governance Specification

Specifying the governance for the SensCity infrastructure consists identifying how the agents can fulfill the strategy objectives using the SensCity resources, and such resources can are abstracted by the governance artifacts.

As described in the previous section, the core platform allows to manage the whole M2M infrastructure. Further more, it is less limited in terms of computing capacity and energy supply than M2M gateways and M2M devices. So, we decide to abstract the core platform components by artifacts. In this section, we describe how we abstract the SensCity platform components with respect to the governance concerns.

The first step is to identify the infrastructure's resources that can use be to fulfill the governance objectives. Such objectives are driven by ETSI TC M2M recommendation and are defined as an horizontal organization, which involves multiple vertical organizations. Then, we define how the agent behave to manage them and to fulfill the strategy requirements using the artifacts, which control the SensCity resources.

In Table 9.2 (page 198), we give an example of such an analysis. In order to (i) fulfill organizational goals, agents (ii) instantiate a plan, which (iii) involves artifact operations, such operations, in turn, (iv) invoke platform components to make the M2M infrastructure effectively compliant with the strategy.

In many case, such resources are sufficient to fulfill the ETSI TC M2M's requirement. For example, the *USP.configurationService* allows to configure and control the client applications' rights regarding the access to the devices and their data.

Nevertheless, it does not enable to specify and control the QoS value of the vertical SLAs attributes. It is particularly important in the case of commands sent to the devices, which implies an important energy and communication cost for the WSAN.

So agents need to set up a governance regulation policy that monitors the client applications' behavior and constraints it to the limit defined in the SLA. To do so, we use a *threshold* policies that will intercept the client applications' requests and let them pass only if the request complies with the SLA.

But, we can also notice that some ETSI TC M2M's requirements are not supported by the platform. That is the case for the *rCBRedeem* objectives, corresponding to the *Compensation and Brokerage* requirement.

However, such requirement is directly supported by the vertical strategy through the contract specific vertical organizations. In fact, such a goal is achieved by the vertical goal *billCompensation*. While the billing operation is not supported by the SensCity platform, this can be easily implemented by the *ApplicationArt* directly or provided by an additional component.

In this section, we have demonstrated that our governance framework is generic enough to be integrated within an existing M2M infrastructure with a minimal effort.

In next section, we detail how the AGaMeMon governance system can be effectively integrated to the SensCity platform, concluding this implementation chapter.

9.3.3 Wrapping Java Beans Components with CArtAgO Artifacts

In order to integrate the AGaMeMon framework into the SensCity architecture, we propose an original hybrid CBSE/MAOP approach. That is, we use governance artifacts to encapsulate the platform's component.

Governance Integration. Figure 9.6 provides a graphical representation of such an hybrid CBSE/MAOP deployment. Within the SensCity platform, legacy components identified by the previous phase are wrapped by *BridgeComponent* components.

Such a *BridgeComponent* component implements the same interface as the *LegacyComponent* component it encapsulates. And so, it can be dynamically deployed instead of the legacy component.

Listing 9.8 gives an example of such a replacement declaration in an XML Java Bean configuration file. In this example, the *eventReceiver* component is replaced by the *DeviceRegisteryBridge* bridge component. When loaded, this bridge component loads a legacy *DeviceRegistery* class instance and encapsulate it with *DeviceArt* artifacts.

Governance Bridge Initialization. The *BridgeComponent* acts as a local proxy for governance artifacts. That is, the governed platform and the governance workspace are located in different virtual nodes. This represents three main advantages: (i) it allows to limit resource use overhead on the platform server generated the governance processes, (ii) several platform instances can be governed at once, and (iii) the bridge component can replace and delegate legacy tasks to different implementations of a component.

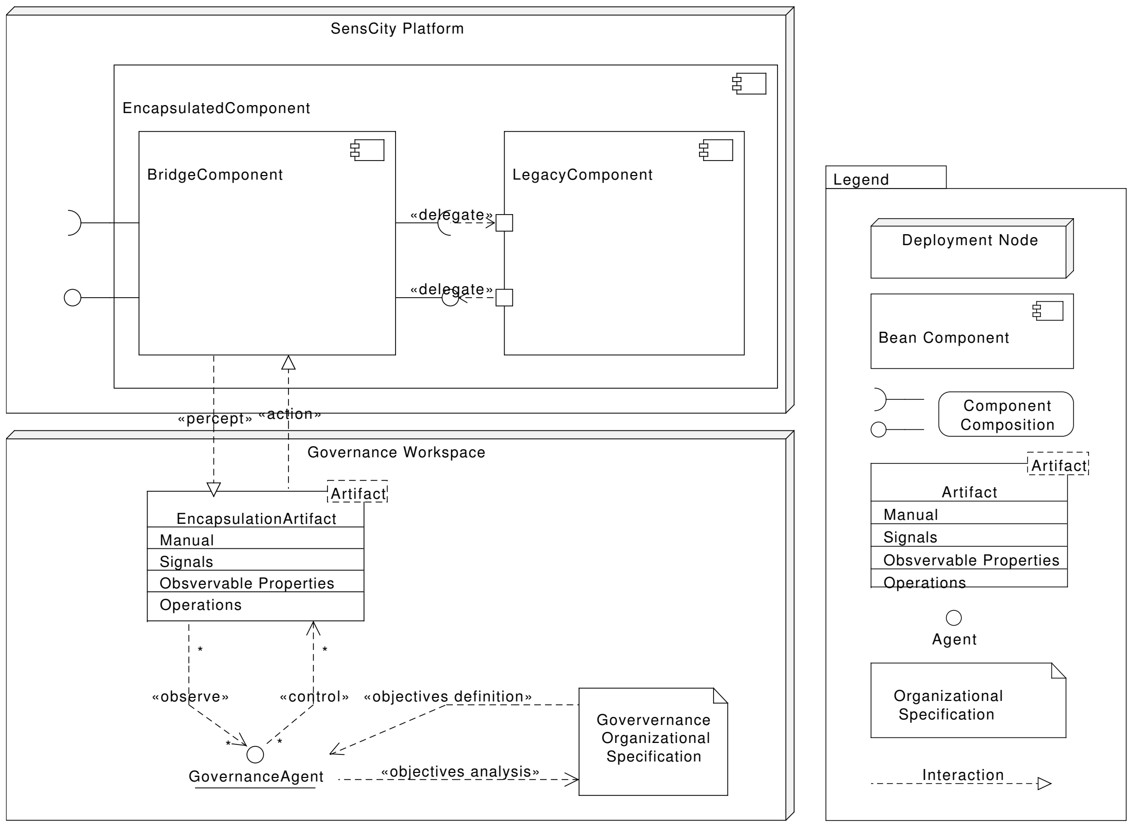


Figure 9.6: Simplified representation of a SensCity Governance deployment.

In Listing 9.9, we show how the *DeviceRegisteryBridge* component is initiated. First, we use the CArtAgO library to link the bridge component with the governance workspace (Line 7). Then, the bridge component loads the legacy SensCity component (Line 11). Finally, it generates *DeviceArt* governance artifacts (Line 16) for each device groups registered in the platform by invoking the legacy component's methods (Line 17).

Governance Perception and Control. Bridge components are used to implement the *controller* and *perception* modules of the governance artifact.

Thanks to such an architecture, the bridge component can monitor a part of the SensCity platform by intercepting the component calls and notifying and/or redirecting such calls to the governance artifact.

To do so, we define specific artifact operations dedicated to the bridge classes' use. For example, the `perceptDevicesAdded` operation allows the bridge component to notify the artifact devices were added to the group (Listing 9.9, Line 22).

Further more, it can invoke the component's methods to configure the M2M infrastructure according to the governance decisions. For example, the `controlRouteSelected` method enables the artifact to update routes to devices (Listing 9.9, Line 40).

9.4 Synthesis

In this chapter, we have presented how we implemented our governance framework AGaMeMon and how it is integrated into the SensCity core platform to govern this M2M infrastructure.

AGaMeMon is implemented using the JaCaMo MAOP framework which is a synergistic combination of the $\mathcal{M}$oise, *Jason* and CArtAgO frameworks, each of them being dedicated to the development of the different MAS dimensions. In order to fit with our governance purpose, we developed governance specific features in order to (i) support the governance processes by the MAS, (ii) enable the agents to analyze and reason about M2M requirements, specified by organizational specifications, and to (iii) embed governance policies with artifacts.

Based on the SensCity project, we provided an example of a governance specification and integration within a real world M2M infrastructure. To do so, we proposed an hybrid **csbe!** (**csbe!**)/MAOP approach, which enabled CArtAgO artifacts to encapsulate Java Beans components.

As the SensCity project was deployed while our work was still at an early stage (2010), we were not able to deploy and validate our approach within a physical deployment. Therefore, we present in the next chapter an experimental evaluation, conducted through simulations.

```
<!-- File: 'com/francetelecom/senscity/device/device-registery.xml' -->
<bean id="deviceRegistery"
      class="fr.emse.agamemnon.bridge.uccp.device.DeviceRegisteryBridge">
  <constructor-arg index="0" type="java.lang.String">
    <value>/fr/emse/agamemnon/bridge/device/device-registery.xml</value>
  </constructor-arg>
  <constructor-arg index="1" type="java.lang.String">
    <value>eventReceiver</value>
  </constructor-arg>
  <constructor-arg index="2" type="java.lang.String">
    <value>fr.emse.agamemnon.artifact.DeviceArt</value>
  </constructor-arg>
  <constructor-arg index="3" type="java.lang.String">
    <value>device</value>
  </constructor-arg>
  <constructor-arg index="4" type="java.lang.String">
    <value>__WORKSPACE_HOST__</value>
  </constructor-arg>
  <constructor-arg index="5" type="java.lang.String">
    <value>__WORKSPACE_NAME__</value>
  </constructor-arg>
</bean>
<!-- EOF -->
<!-- File: '/fr/emse/agamemnon/bridge/device/device-registery.xml' -->
<bean id="deviceRegistery"
    class="com.francetelecom.senscity.device.DeviceRegistery" />
<!-- EOF -->
```

Listing 9.8: XML declaration of a Java-Bean Encapsulation Artifact.

```java
public class DeviceRegisteryBridge extends cartago.util.agent.Agent
                      implements com.francetelecom.senscity.IDeviceRegistery {
 /* ... */
 public DeviceRegisteryBridge(String beanID, String beanFile, String artifactType,
                              String workspaceHost, String workspaceName) {
   // Link Bridge to the Governance Workspace
   super("DeviceRegisteryBridge", workspaceName, workspaceHost);
   // Retrieve Legacy Component
   XmlBeanFactory bf = new XmlBeanFactory(new ClassPathResource(beanFile,
                                                                getClass()));
   this.legacy = IDeviceRegistery.class.cast(bf.getBean(beanID));
   // Generate Governance Artifact
   this.deviceArtifacts = new HashMap<String,ArtifactID >();
   this.initArtifacts(artifactType);
 }
 public void initArtifacts(String artifactType) {
   for(DeviceGroup dg : this.legacy.getDeviceGroupList()) {
     ArtifactID govArtifact = this.generateArtifact(dg.getID(), artifactType);
     if(govArtifact != null) {
       Object[] deviceIDs = {this.getDeviceIDList(dg.getDeviceList())};
       // Use governance monitoring Percept to added the group's devices
       doAction(govArtifact, new Op("perceptDevicesAdded", deviceIDs));
       this.deviceArtifacts.put(dg.getID(),govArtifact);
     }
   }
 }
 // Generate DeviceArt artifacts for the device groups
 public ArtifactID generateArtifact(String deviceGroupID, String artifactType) {
   ArtifactID aid;
   Object[] params = {this}; // To enable Bridge direct invocation by the Artifact
   try { aid = makeArtifact(deviceGroupID, artifactType, params);
   catch (CartagoException e) {
     try {
       aid = lookupArtifact(deviceGroupID);
       doAction(aid, new Op("registerBridge", params));
     } catch  (CartagoException e) { return null; }}
   return aid;
 }
 // Governance Controller to configure the M2M infrastructure
 public void controlRouteSelected(String artifactID, String routeID) {
   for(Device d : this.legacy.getDeviceGroup(artifactID).getDeviceList())
     this.legacy.updateRouting(deviceID, this.getRouteToDevice(deviceID,routeID));
 }
}
```

Listing 9.9: *DeviceRegisteryBridge* component initialization.

CHAPTER 10

M2M Governance Simulation

Contents

In the previous chapter, we have described how we can implement our governance proposal using the JaCaMo framework and deploy it on a real existing M2M infrastructure (SensCity). In this chapter, we now evaluate our proposal in order to validate our approach and discuss it.

Based on a use-case scenario, we simulate an M2M infrastructure and run our framework to demonstrate our approach with respect to M2M governance issues. These experiments will allow us to evaluate and discuss the benefits of our proposal.

In this chapter, we first introduce the use case scenario and the experimentation protocol, in Section 10.1. Then, we present and analyze two simulation experiments. First, in Section 10.2, we show a vertical SLA refinement initiated by the client application's needs. Then, we describe how devices' constraints can be taken into account to regulate the M2M infrastructure and refinement the vertical refinements, in Section 10.3.

Finally, in Section 10.4, we synthesize these experiments to validate and discuss the different aspects of our approach.

10.1 Use Case scenario and Experimentation Deployment

In this section, we define an experiment use-case scenario that exhibits M2M governance issues. Based on this scenario, we describe the simulation protocol we follow for each experiment.

10.1.1 Use-Case Scenario

Let us consider an activity monitored by sensors which increases and decreases depending on the time period. For example, the activity of parking spaces may evolve during the day: in the morning, parking spaces are freed in residential area and cars are parking near working or shopping areas, not much changes are observed during the day until the evening, then the cars go back to the residential areas.

We assume the parking sensors can be reconfigured at run-time. Hence, different Non Functional Requirement (NFR) can be defined and deployed at run-time. And so, the data collection requirements can be adapted to the end-user demand, by the client application, but also to the devices' constraints.

The corresponding governance should manage such an SLA definition to validate, enable and monitor it within the M2M infrastructure. To do so, the following governance aspects should be treated:

1. SLA must be defined according to the client application requirements and be validated and deployed by the governance system.
2. The M2M infrastructure must be monitored to check the validated SLA.
3. M2M devices' constraints must be taken into account in order to keep the infrastructure's viability.

10.1.2 Experimentation Protocol

With these objectives in mind, we now define how we are conducting our experiments. For each experiment, we define a governance aspect we focus on. According to this objective, we define the corresponding governance specifications, to show the governance definition and deployment effort.

Then, we provide the simulation results and evaluate them, with respect to the M2M governance perspective. We analyze and discuss the governance contribution.

Simulation Definition. Based on our scenario, we build a simulation model focused on the end-to-end vertical management. Therefore, we simplify the core platform's and gateways' tasks to message transmission. For the same reason, we do not use WSAN repeaters, avoiding the recruitment phase, which is not relevant for our purpose.

In order to simulate the long term impact of the governance over the M2M infrastructure, each step of the simulator represents a minute in the real world. Each simulation is run during a 6 years simulated duration.

Our simulator is composed of (i) the environmental simulator, (ii) the M2M infrastructure and (iii) the governance layer. Hence, we define the following simulator components:

Parking Dynamic Simulator. It models the parking places dynamics based on a simple car driver behavior: a pic hour is defined given the parking location –i.e. "Business" or "Residential" area– during which many places left or occupied, and few changes occurs during the rest of the day.

Parking Sensors. They are WSAN devices able to sense a parking place occupation. It can be configured to report each parking status modification (`ON_EVENT` mode), or to report such a status regularly (`NOTIFY` mode) given a frequency parameter. Such sensors are grouped together according to the type of area they are located in.

M2M Gateways. They transmit configuration commands and notification messages to and from the parking sensors.

Core Platform. It provides a communication interface with the M2M gateways and a client application interface to register and subscribe to the sensors, with an SLA proposition. Application requests are intercepted by the governance bridge layer to be validated by our governance layer. The core platform maintains a subscription base, containing those validated by the governance layer.

Client Application. It subscribes to each parking sensor groups to collect the whole parking places state. We compare five different client application SLA profiles, each one using a different requirement specification.

Table 10.1 summarizes such profiles with the corresponding NFR. The `ON_EVENT` SLA profile corresponds to a subscription to each event notification, the `NOTIF(30)`, `NOTIF(60)` and `NOTIF(120)` SLA profiles correspond periodic state notifications, with different frequencies, and the `ADAPTIVE` SLA profile adapts its subscriptions requirements to the drivers' demand.

Governance Layer. It controls the M2M infrastructure to fulfill the vertical requirements. In order to avoid issues due to the asynchronous behavior of JaCaMo and the simulation speed, we have implemented a simplified and lightweight version of our governance model. The goal here is to focus on each experiment objective in order to demonstrate the benefits of our approach.

Profile	*Requirements*
`ON_EVENT`	`ON_EVENT` notifications
`NOTIF(30)`	`NOTIFY` notifications, every 30 minutes
`NOTIF(60)`	`NOTIFY` notifications, every 60 minutes
`NOTIF(120)`	`NOTIFY` notifications, every 120 minutes
`ADAPTIVE`	`ON_EVENT` notifications, during pic hours
	`NOTIFY` notifications, every 120 minutes, otherwise

Table 10.1: Non Functional Requirements corresponding to the *Application Vertical Refinement* experiment SLA profiles.

Performance Measurements. Performance of each SLA profile is evaluated along two metrics: application's *information accuracy* and devices' *battery lifetime*. To produce such measurements, we run our simulation with two group of 10,000 devices each. Each run last the equivalent 6 years, that is the worst device lifetime –i.e. `ON_EVENT` profile. The time unit used in our simulator is a minute per cycle.

Information Accuracy. It refers to the accuracy of information about the parking places the application has, with respect to the real state of the corresponding places. That is, we measure the percentage of the application's correct information along the day. Such a measure is performed every 10 minutes. Then it is computed as an average value for the whole simulation over the day.

Device lifetime. As stated in Section 1.2.2.1, it is a key issue regarding urban M2M infrastructure. Therefore, it is important to compare the impact of the different NFR on this criterion. In our experiments, we measure battery state for all devices along the simulation once a day.

10.2 Application Vertical SLA Refinement

In this section, we focus on the vertical SLA definition and deployment process. It is initiated by the M2M client application, so that it can provide an efficient service to the end-users.

10.2.1 Governance Definition

In this experiment, governance is involved to validate the SLA proposition and deploy the corresponding infrastructure configuration. Two governance processes, proposed in Section 7.1.2, are demonstrated: (i) adaptation to M2M entities registration, and (ii) SLA validation and deployment.

When a client application or M2M devices register to the core platform, (i) the governance bridge notifies the corresponding artifact –respectively, the *AppArt* and *DevArt* artifacts– which, in turn, (ii) notifies the governance agents. Finally, (iii) dedicated governance agents are instantiated to govern the registered M2M entities.

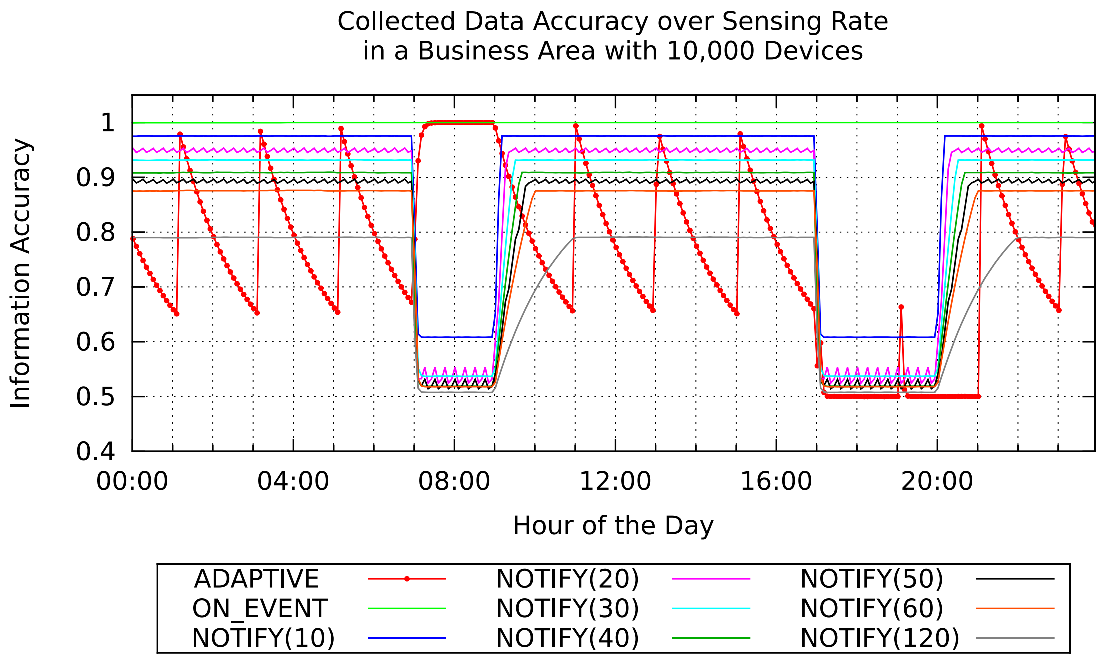


Figure 10.1: Data Collection Accuracy comparison between SLA Profiles.

When a client application (i) requests a subscription to a group of devices, with an SLA proposal corresponding to its profile, the governance bridge deployed on the core platform (ii) intercepts the request and redirects it to the governance layer via the *AppArt* artifact, which (iii) sends a signal to the *appAg* agent responsible for this application. Such an agent (iv) instantiates a vertical organization with the SLA proposal and (v) recruits the other governance agents to participate in the vertical governance. (vi) Each agent evaluates the SLA proposal feasibility and (vii) commits to the vertical organization if it validates it. Finally, when all the roles are committed to governance agents, (viii) the *appAg* agent start the social scheme which makes the other agents (ix) configuring the M2M infrastructure accordingly. Let us note that, in the case of the `ADAPTIVE` profile, such a process is performed twice a day, while for others only one validation is performed, at the beginning of the simulation.

10.2.2 Simulation Results and Analysis

After running each scenario, the results obtained are given in Figure 10.1 (*Information Accuracy*) and Figure 10.2 (*Device Lifetime*).

Information Accuracy. Figure 10.1 presents the accuracy of collected information in a business area for every SLA profiles. We divide our analysis in two phases: (i) high demand period –i.e. between 7 AM and 9 AM– and (ii) low demand period –i.e. before 7 AM and after 9 AM. Although, we can distinguish a third period –between 5 AM and 8 AM– it is not relevant for our purpose as it corresponds to the period when cars massively leave parking places, and so, we can consider the demand is low in this area.

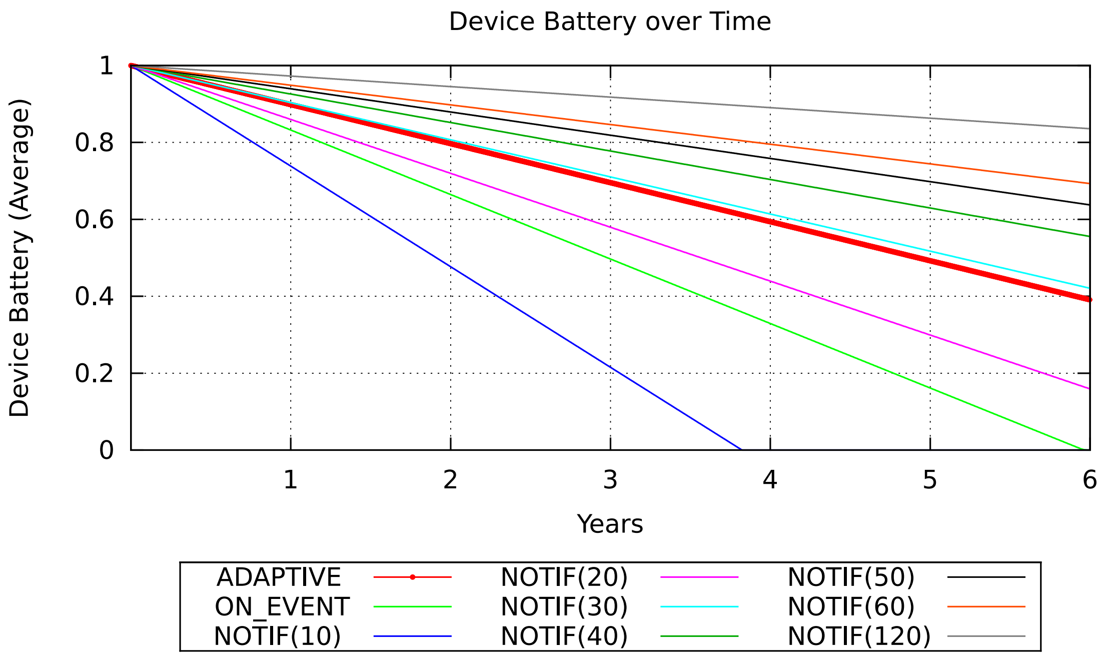


Figure 10.2: Device Battery Consumption comparison between SLA Profiles.

`NOTIFY(X)` SLA profiles all show acceptable performance during low demand hours: from 78% for the lowest frequency notification profile –i.e. `NOTIF(120)`– to 97% for the highest frequency notification profile –i.e. `NOTIF(10)`. Nevertheless, accuracy drops down when environment dynamics get more important – from 52% to 61%, respectively– which corresponds to the high user demand period.

As one could expect, the `ON_EVENT` profile shows the best accuracy performance, as every change in the environment is directly notified to the application. Although, a 100% accuracy is not necessary when demand is low, that is the major part of the day, and consume much M2M resources, especially devices' battery, as we will see further.

Finally, the `ADAPTIVE` profile gets reasonable results during low demand period: 73% average and a minimum of 64%. In addition, it is reactive to the user demand and gets an accuracy of 97% within 10 minutes.

Device Lifetime. Figure 10.2 shows the average battery consumption resulting from each SLA profile's requirements. Due to our simplified battery model, consumption is too linear to be realistic. For example, it does not take into account the battery worsening.

Though, these results enable us to compare the different SLA profiles. We can compute each profile's consumption model $c(t)$ as defined in Equation (10.1), where C is a consumption coefficient over time. Based on this, we can estimate a theoretical device lifetime, as reported in Table 10.2.

$$\frac{d}{dt}c(t) = \frac{t}{C} \tag{10.1}$$

Profile	*ADAPTIVE*	*ON_EVENT*	*NOTIFY*						
			10	20	30	40	50	60	120
C	2.78 $\times 10^{-4}$	4.57 $\times 10^{-4}$	7.16 $\times 10^{-4}$	3.83 $\times 10^{-4}$	2.64 $\times 10^{-4}$	2.03 $\times 10^{-4}$	1.65 $\times 10^{-4}$	1.4 $\times 10^{-4}$	7.47 $\times 10^{-5}$
Lifetime	3602	2190	1396	2608	3788	4937	6059	7159	13383

Table 10.2: Theoretical Device Lifetime (in days) estimation based on simulation results.

Best device lifetimes are obtained with low frequency notification profiles, i.e. `NOTIF(60)` and `NOTIF(120)`, which are greater than 19 and 36 years, respectively. Yet, the `NOTIF(30)` is still acceptable as it is greater than 10 years. Nevertheless, higher notification frequencies lead to a low device lifetime.

The `ON_EVENT` profile leads to a device lifetime of less than 6 years. Thus, it pays the price for its accuracy performance.

Finally, the `ADAPTIVE` profile's consumption is similar to the `NOTIF(30)` one, although the device lifetime is a bit inferior to 10 years –i.e. 9.75 years. Hence, the `ADAPTIVE` profile offers an interesting trade-off between data collection performance and device lifetime.

Conclusion. This experiment demonstrates the benefits of an explicit SLA declaration and governance management. In fact, doing this allows each stakeholder to keep control over its own requirements. Thus, stakeholders can come to trade-offs satisfying each one interests.

In our scenario, the application owner stakeholder might seek for the following SLA profiles: 1) `ON_EVENT`, 2) `ADAPTIVE` and 3) `NOTIF(10)`, by order of preference. While the parking sensor provider might prefer *NOTIFY* profiles with a frequency above 1 message each 30 minutes, or the `ADAPTIVE` profile, to reduce the maintenance cost. Hence, the `ADAPTIVE` SLA profile offers the most satisfying trade-off between energy consumption and information accuracy.

In this experiment, we used a relatively passive governance that only validates the SLA, configures and monitors the M2M infrastructure, accordingly. Next experiment will show how we can automate such an SLA negotiation and refinement, using the agents' regulation and reorganization plans.

10.3 Device Regulation and Vertical SLA Refinement

In the previous section, the governance was driven by the application's interests. We showed that adapting requirements to the environment dynamics provided an interested trade-off with respect to (i) the accuracy of collected data and (ii) devices' lifetime. In this section, we focus on the vertical regulation and refinement process driven by M2M devices' constraints and performed by the *devAg* governance agents.

10.3.1 Governance Definition

In this experiment, governance is involved to improve devices' lifetime. Three governance processes are demonstrated: (i) dynamic vertical SLAs refinement (Section 7.1.2), (ii) dynamic reconfiguration of the M2M infrastructure (Sections 8.1.2 and 8.3.1), and (iii) agents' scale-out/down strategies (Section 7.1.3).

In order to highlight the battery lifetime issue in this scenario, we take into account the battery degradation. That is, while time goes batteries' leakage will grow. Hence, an old battery will tend to consume more than a new one. In addition, we set 30% of the devices with extra leaking batteries, in order to simulate the heterogeneity of devices' battery consumption.

In order to handle such an issue, we define the governance as follows.

When M2M devices' lifetime is detected to be *critical* (see next paragraph), (i) the device governance bridge notifies the corresponding *DevArt* artifacts, which (ii) alert the device agents. Agents (iii) decide to separate the critical devices from the others when the number of such devices is important enough to create a new device group –set by a `CONSISTENCY` parameter (30 in our simulations). If agents consider it worth creating a new device group, (iv) they use the artifact to generate such a group, called the "degraded device group", and (v) starts the *Device Lifetime Management* policies to automatically migrate critical devices. To govern the new device group, (vi) a new *DevAg* agent is created. (vii) Such an agent proposes a new SLA –with lower NFR parameters– to the *AppAg* and *InfAg* agents. When validated, (viii) the new SLA is deployed as a new subscription between the client application and the new device group.

If a device group is too small –i.e. contains a number of devices inferior to the `CONSISTENCY` parameter–, the group should be merged with an other one in order to avoid the multiplicity of device groups. This is detected by the device artifacts which (i) move the remain devices to the "degraded device group" and (ii) alerts the agents. The concerned *DevAg* agent (iii) terminates all subscriptions to this group by stopping the corresponding organizations. When this is done, (iv) it deletes the device group from the infrastructure registry via the artifact and (v) disposes it. Finally, (vi) the agent terminates itself, realizing a scale-down in the governance MAS.

Critical state of devices criteria, the device group *consistency* number and the "degraded" QoS values are parameters which are to be defined by the devices' owner. In our simulation scenario, we set those value arbitrary, as follows:

- The `CONSISTENCY` value is set to 30 devices.
- A devices is considered critical when its battery consumption is below 98% of the average consumption and when 50% of the battery is been consumed.
- The degraded QoS for critical is set to one notification per hour. Table 10.3 summarizes NFRs corresponding to the initial (`ADAPTIVE`) and the generated (`DEGRADED`) SLA profiles.

Profile	*Requirements*
`ADAPTIVE`	`ON_EVENT` notifications, during pic hours
	`NOTIFY` notifications, every 120 minutes, otherwise
`DEGRADED`	`NOTIFY` notifications, every 60 minutes

Table 10.3: Non Functional Requirements corresponding to the *Device Vertical Refinement* experiment SLA profiles.

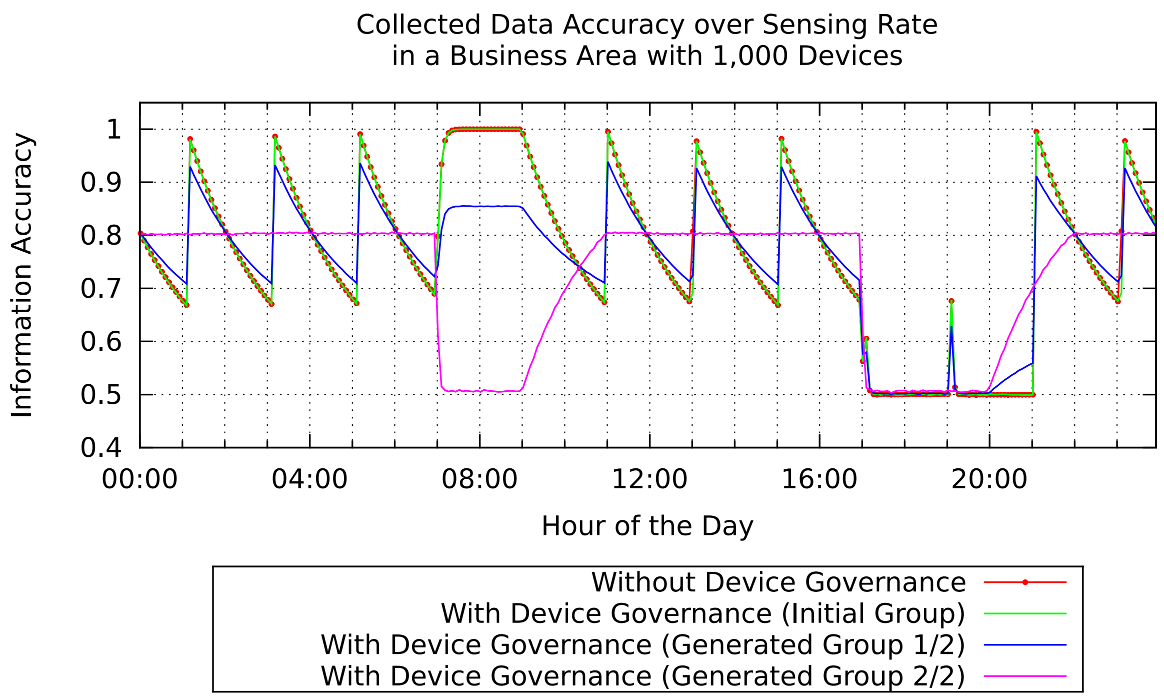


Figure 10.3: Data Collection Accuracy comparison with respect to the Device Driven Governance.

10.3.2 Simulation Results and Analysis

Given the scenario and simulation parameters previously defined, we analyze, in this section, results provided by our simulation runs with 1,000 devices. As the `ADAPTIVE` SLA profile provided the best trade-off between information accuracy and devices' lifetime, we use this profile to compare performances. On the one hand, the new scenario is deployed using the `ADAPTIVE` profile and no additional governance. On the other hand, we deploy the device driven governance in order to improve devices' lifetime, using the same initial SLA profile.

From Figure 10.4, we can observe two main events occurring during the simulation progress: (i) a degraded group is generated on day 825 (`S`), then, critical devices are progressively migrated to this group (`S'`, `S"`), until (ii) day 8190 (`M`), when all devices remaining in the initial group have reached the 50% battery threshold and the two groups are merged into the second one.

Information Accuracy. Figure 10.3 presents the average accuracy of collected data in a business area. The first graph –red line with dots– represents the simulation which does not use the device driven governance, while the three others result from the device lifetime governance, respectively, before critical devices are migrated to a new group –green line– with a lower message frequency QoS, while some devices are migrated to the new group and other still remain in the initial one –blue line– and after all devices were migrated to the degraded group –pink line.

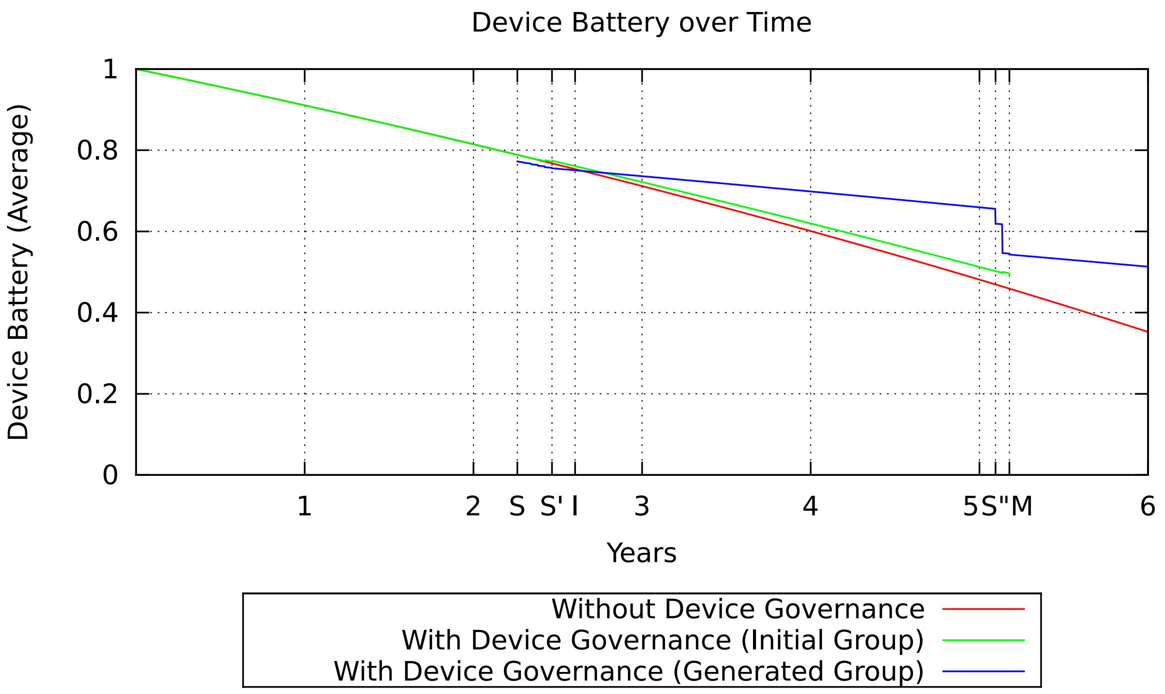


Figure 10.4: Device Battery Consumption comparison.

Considering the business area where the devices are located, the low demand period can be delimited before 7 AM and after 9 AM. During this period all cases have a similar accuracy, oscillating around 80%. It is sensibly better after critical devices are migrated to the new device group, because such devices send notifications every 60 minutes, while with the initial profile they send notifications every 120 minutes.

During the high demand period –i.e. between 7 AM and 9 AM– differences are more significant. Until a degraded group is created (or when no degraded group is created, in case 1), accuracy quickly reaches 100%, thanks to the `ON_EVENT` notifications. But concerning critical devices, accuracy drops down to 50%. Nevertheless, it still provides an 85% average accuracy during a period of nearly 3 years, while 70% devices still have more than 50% of their battery.

Device Lifetime. Figure 10.4 describes the average battery consumption during the simulation. The first curve (red line) shows battery consumption for the `ADAPTIVE` SLA profile without device governance, while the two others are generated when the device governance is activated. The second curve (green line) represents the initial device group's average battery consumption and the last one (blue line) represents the generated group with degraded SLA.

These results show a significant battery consumption improvement brought by the governance action. In fact, starting from day 900 (`I`) critical devices' average battery state keeps above the two others. As a result, the average battery state in the initial group gets higher than in the case of the *ADAPTIVE* profile without device governance.

10.4 Simulation Synthesis

In this chapter, we have presented an empirical evaluation of our M2M governance proposal, based on simulations.

With our first simulation pack, we have demonstrated the benefits of using an explicit SLA to govern the M2M infrastructure. Such an SLA allows the client application to fit its usage of the M2M infrastructure to its needs, such as the end-user demand. Hence, the `ADAPTIVE` SLA profile provided the best trade-off between the accuracy of collected information and the device's energy consumption.

With our last simulation pack, we have demonstrated that it was possible to improve such a trade-off. As the governance system is able to analyse the state of the M2M devices, it can provide a finer grained SLA definition. In fact, in our simulations, critical devices were isolated from the others by the governance layer, which defined a new SLA in order to improve devices' lifetime.

Such experiments allow us to validate our proposal as a promising approach for governing M2M infrastructure, although our model was simplified for the simulations' purpose. Through this means, we have demonstrated the benefits of the thesis we defend in this manuscript, that is (i) how M2M governance can be specified in order to take into account each stakeholder's requirements and M2M resources constraints, and (ii) how it can be automated in order to improve the control and the management of the M2M resources life cycle.

CONCLUSION

In Parts I, II and III, we presented the work we realized in this thesis. This last chapter summarizes the main ideas of this dissertation and draws perspectives for further research based on our contributions. We proposed an Machine-to-Machine (M2M) governance framework, named AGAMEMON (*Adaptive Governance MAS for M2M*), based on a Multi-Agent Oriented Programming (MAOP) which allows to automate the governance processes.

We first remind the context of this thesis, before synthesizing our contributions regarding the governance of M2M systems. Limits of this work and further research tracks are discussed at the end of this conclusion.

Context Reminder

M2M systems are N-tier infrastructures connecting Wireless Sensor and Actuator Network (WSAN) devices to value added M2M services using intermediate servers, namely, the M2M core platform and M2M gateways. To build such an M2M infrastructure, we can identify two approaches: (i) vertical and (ii) horizontal M2M infrastructures.

On the one hand, a vertical approach consists of a closed infrastructure where all M2M entities are provided and maintained by a single or a limited number of stakeholders. While this guarantees a better control of the infrastructure, it also limits the size of such an M2M system as it requires many competences and could lead to expensive infrastructure costs.

On the other hand, an horizontal approach aims at sharing the maintenance and costs among several stakeholders in an open infrastructure. That is, each stakeholder provides M2M entities falling into its field of competences: (i) device providers set up M2M sensors and actuators, (ii) Telecommunication Operators (Telcos) provide network connectivity and WSAN repeaters, and (iii) software companies develop value-added services to the customers by subscribing to the M2M devices.

Nevertheless, horizontal M2M deployments raise important issues regarding the constrained nature of M2M resources. In fact, WSAN traffic and M2M devices lifetime are critical in order to maintain an acceptable Quality of Service (QoS) level and maintenance costs. Such issues are enforced by the open nature of an horizontal infrastructure, making its size potentially infinite, and so, raising scalability issues.

Therefore, horizontal M2M infrastructures raise the problem of a governance process adapted to M2M specific constraints. While, standardization efforts facilitate interoperability between M2M stakeholders, we identified remaining issues with respect to M2M concerns.

1. **Heterogeneity of application requirements** because of the various number of stakeholders sharing the M2M infrastructure. In fact, different client applications have different requirements in terms of resources to use, but also in terms of QoS.

2. **Device lifetime management** induced by the constrained nature of M2M devices and WSAN. This is strengthened by the heterogeneity of client applications and QoS requirements, which could lead to a lack of device control. This is of major importance as human intervention is expensive –e.g. to replace batteries.

In order to determine how to govern M2M systems, we reviewed the literature for classical Information Technology (IT) governance and Service Oriented Architecture (SOA) governance. As a matter of fact, M2M systems are too recent to define good practices, but many governance concepts can be applied to the M2M.

Such a study revealed that governance can be defined by several governance layers from global business objective to finer grained resource use specifications. We formulate these layers as follows. First, the *governance strategy* layer specifies the global system's requirements based on business objectives. Then, the *governance tactic* defines intermediate objectives and means to fulfill the global requirements. Finally, the *governance policy* layer controls each resource's usage according to the upper levels objectives.

Given the dynamics and the scale of M2M systems, we stated that such a governance process should be automated. After reviewing several adaptive and control systems approaches, our survey revealed that MAOP provided interested means for automating the governance processes.

Therefore, we propose in this thesis an innovative approach for governing horizontal M2M infrastructures, based on MAOP to encapsulate the different governance layers. In the next section, we summarize such contributions provided by this thesis.

Contribution Synthesis

The main contributions of this thesis are: (i) a proposal of an M2M governance model, and (ii) an MAOP approach to support M2M governance processes. Then, we also contribute to (iii) an innovative multi-organizational design for separating the horizontal and vertical concerns, and (iv) a proof-of-concept integration with an existing M2M infrastructure using a novel CBSE/MAOP hybrid system.

M2M Governance Model

Many work discuss several M2M management issues, such as N-tier infrastructure, M2M device management and discovering or scalability. These works have a specific focus on technical solutions for M2M infrastructures. Nevertheless, to our knowledge, they do not address the problem of an M2M specific governance model. Such a lack of M2M governance explains why industrial stakeholders are reluctant in deploying horizontal M2M infrastructures.

Our first contribution is to provide a base model for governing horizontal M2M infrastructure. This governance model takes its principals from IT and SOA governance to which add the distinction between the horizontal and vertical governance.

That is, business objectives drive the *governance strategy* definition which specifies the overall system requirements. We define the strategy in two dimensions: (i) the *horizontal* dimension expresses the overall M2M infrastructure objectives (following the ETSI TC M2M recommendations), while (ii) the *vertical* dimension expresses the end-to-end communication between specific M2M entities, corresponding to virtual contract between the stakeholders.

From such requirements, the *governance tactics* layer defines intermediate objectives with respect to the different stakeholders objectives: (i) the client application, (ii) the device provider and (iii) the Telecommunication Operator (Telco). Such tactics balance the different stakeholders' interests with the overall functioning of the M2M infrastructure. This allows to determine which vertical contracts can be deployed and those which are no longer bearable for the M2M infrastructure.

To apply the tactics decisions, we define *governance policies* to regulate the usage of each resources (i.e. M2M entities). Finally, the *governance refinement* aims at continuously adapt all these objectives based on the governance performance and the M2M infrastructure dynamics.

MAOP Approach for M2M Governance

Our second contribution is the use of a Multi-Agent System (MAS) as a means to embed the governance process. To do so, we adopted an MAOP approach to benefit from the different MAS dimensions' abstraction levels. Hence, we developed an *Adaptive Governance MAS for M2M* (AGAMEMON) using the JaCaMo MAOP framework.

Other approaches could have been used, such as autonomic computing. Autonomic computing is widely spread in the network management community and offers different levels of abstraction. Nevertheless, autonomic managers usually have a more specific focus on self-* concerns rather than a governance focus. As a matter of fact, *infrastructure* artifacts' policies could embed autonomic managers. Though, artifacts offer more governance functionalities (see below).

Following such an MAOP approach, we defined the governance strategy as an Organizational Specification (OS), which makes the agents aware of such objectives. We divided the organization into an horizontal OS and a vertical OS template.

The horizontal OS translate the ETSI TC M2M capabilities and functionalities into an agent organization, while the vertical OS represents the structure and the terms of a vertical M2M contract. This allows us to dynamically add, maintain and remove vertical contracts at run-time, and facilitate the separation of concerns.

Using BDI agents, we build a tactic layer that is able to take into account both the horizontal and vertical objectives. We defined three types of agents –i.e. *Application*, *Device* and *Infrastructure* agents– representing the different stakeholders' point of view –respectively, the client application, the device provider and the Telco stakeholders.

For each type of agents, we define plans that binds the horizontal and vertical organizations and apply such objectives to the M2M infrastructure. They use the artifact layer to define different means to regulate the M2M activity and to enforce the fulfillment of the requirements.

Artifacts provide an abstraction of the M2M infrastructure and a support for embedding the governance policies.

Artifacts are interfaced with the infrastructure's M2M entities so that they can monitor and control them. In turn, they offer an interface to the agents to evaluate the functioning of the M2M infrastructure by means of artifact signals and properties. Artifact operations allows the agents to control the artifact's behavior and, indirectly, the M2M infrastructure one.

Thus, agents can activate and deactivate regulation policies dynamically and evaluate their performance. Such policies can be tuned to enforce or relax some requirements. In addition, artifacts can be linked with other artifacts so that they can coordinate the governance of different M2M entities.

Finally, interactions allow us to define adaptation and reorganization processes for each MAS layer to enable the governance refinement. Vertical OS' parameters can be modified in order to adapt the QoS attributes, and so, balance them with the infrastructure constraints. We do not reorganize the horizontal OS, but agents can reorganize the Organizational Entity (OE), by the means of coalitions, so that the governance system scales-out to fits to the M2M dynamics. Governance policies enable the artifacts to adapt the M2M infrastructure, creating and terminating M2M entities. In turn, artifacts are automatically generated and disposed in function of the M2M entities life-cycle.

Additional Contributions

Beside our main contributions, this thesis also addresses innovative approaches, used to enable the M2M governance, but which could also contribute to other software engineering and Multi-Agent System research domains, namely, (i) a multi-organization design for open MAS, and (ii) an hybrid CBSE/MAOP integration. Following, we briefly synthesize the benefits of such contributions.

Scalability Governance. In Section 1.3, we synthesized the literature about scalability management to proposed our own definition of a scalability governance (Definition 3). We stated that scalability can be managed by a governance process which constantly adapts the system's properties, based on trade-offs, to able the system to scale.

...

Multi-Organization design for Open MAS. As a means to separate the horizontal and vertical strategy concerns, we propose to split the multi-agent organization into several smaller ones. This facilitate the specification and the maintenance of such organizations compared to a single large-scale OS.

Such a multi-organizational design is of great interest not only for M2M governance, but also regarding other open and decentralized systems involving multiple stakeholders. In fact, SOA and cloud systems, for example, usually involve third party systems which in turn can be involved in other collaborative systems. Using multiple organization would allow designers to naturally integrate their system with several different collaboration schemes, using agents to balance them with respect to their own requirements.

Though, while it reduces the complexity of each organization, it raises the problem of coordinating them. In this thesis, we define such a coordination at the agent level. Nevertheless, it does not respond to all the possible outcomes. We discuss these aspects further in this conclusion.

Hybrid CBSE/MAOP Integration. In order to integrate the AGAMEMON governance framework with the SensCity core platform, we used an hybrid CBSE/MAOP approach. We encapsulated SensCity components with bridge components which implement the governance controller and perception modules.

Such a design allows to combine Component-Based Software Engineering (CBSE) for programming the business logic and MAS for programming the intelligence and adaptation logic, and so, provide a better separation of concerns. In fact, as mentioned in the State-of-the-Art part, the Agents & Artifacts (A&A) paradigm offers a first-class abstraction of the internal and external (to the MAS) resources. In this thesis, we combined JaCaMo artifacts together with Java Beans components to realize an adaptive control loop, enabling the artifacts to plug and unplug components dynamically.

Discussion and Further Work

In the previous section, we synthesized the contributions of this thesis. We now discuss the limits of our work in order to identify further research tracks that, in our opinion, should be explored.

M2M Governance Model Maturity

Our thesis contribute to fill a gap in the design of horizontal M2M systems by providing an M2M specific governance framework. Although, this is still a preliminary work. In fact, many aspects could not be addressed. Hence, such a framework should be enriched to take into account other M2M governance concerns.

One of the main reasons why this framework could not be exhaustive is that M2M is still a new technology. Though it is a growing field of interests, only few deployments are already running and many of them are still Research & Development or proof-of-concept projects. This is even more the case for horizontal M2M infrastructures.

IT and SOA governance frameworks also comes with *best practices* notes to drive the infrastructure design and requirement specifications. Such best practices are issued from many system administrators' experience, collected and agglomerated together. Though, our approach is mainly based on the experience of the SensCity project. Further more, we treated only the main SensCity vertical schemes (*Data Collection* and *Command Sending*), but left some others.

As a consequence, our governance model should probably undergo several improvements. Nevertheless, we believe our approach provides a good basis for such improvements as MAOP offers flexibility and a clear separation of concerns. Though, we can already point out some improvements that could be addressed:

Since, we did not treat all the possible vertical schemes, additional ones should be defined. We cannot provide an exhaustive list of such schemes, however we can mention one from the SensCity use cases: *Local Device Control*. That is, the possibility to trigger commands locally –e.g. by the M2M gateway– to actuators when some events are sensed to provide faster reactions, instead of (i) sending a notification to the client application, so that (ii) it sends the commands to the actuators, through the whole M2M infrastructure.

Using multi-agent reinforcement learning techniques, the tactic agents could be able to remember the consequences of the different tactics and decisions applied. Doing this, the agents could build a "best practice" base, for later use, analysis and improvements.

Interdependency between Vertical SLAs

Another aspect to be addressed is the treatment of dependencies and incompatibilities between several vertical contracts. This raises several issues which were not addressed in this thesis.

As mentioned previously, it raises some issues regarding the multi-organizational design. In this thesis, we used the agent's knowledge so that it could bind horizontal and vertical roles, without any organizational constraints, except for the *SLA Validation* scheme.

Nevertheless, such an agent's knowledge clearly expresses an organizational constraints. This could be expressed as a norm, for example: "role r_H is obliged to commit to r_V", where r_H and r_V denoted roles in the horizontal and the vertical organizations, respectively.

Thus, such a design should undergo deeper investigation in order to provide a better inter-organization formalization.

In addition, the agent's reasoning towards such multiple organizations would also require further research. In fact, balancing constraints from several organization, requires the agents to be able to prioritize these constraints and to establish trade-offs, individually but also collectively.

For example, in this dissertation, we simply defined the priority to the first vertical contract set up, in case of conflicts. That is, new Service Level Agreement (SLA) proposals are validated if it does not disturb the previous one. One may prefer to define such a priority on different basis –e.g. financial benefits, type of institutions.

Automatic SLA Negotiation

Related to the previous point is the need for an automatic SLA negotiation. In the AGAMEMON framework, the QoS attributes are defined manually by the human stakeholders and inserted into the OS. Though such a definition could be automated and negotiated by the agents. This would require additional features to our proposal.

We believe the governance framework should provide means to define each M2M entity's profile. In AGAMEMON, we defined different types of agents representing the different concerns for each types of M2M stakeholder. Though, different stakeholder of the same type, or even different M2M entities belonging to the same stakeholder, may have different objectives and constraints. Thus, an explicit specification of such individual requirements would enable the agents to negotiate the vertical SLAs.

PUBLICATIONS

This work was partly published in the following peer-reviewed workshops and conferences. Following is a list of such publications.

- Persson, C., Picard, G., Ramparany, F., and Boissier, O. (2014). A Multi-Agent based Governance of Machine-to-Machine Systems. In *International Workshop on Web Intelligence and Smart Sensing (IWWISS'14)*. ACM Digital Library.
- Persson, C., Picard, G., Ramparany, F., and Boissier, O. (2012). Multi-Agent Based Governance of Machine-to-Machine Systems. In Cossentino, M., Kaisers, M., Tuyls, K., and Weiss, G., editors, *9th European Workshop (EUMAS 2011), Revised Selected Papers*, volume 7541 of *Lecture Notes in Computer Science (LNCS)*, pages 205–220. Springer.
- Bilal, M., Persson, C., Ramparany, F., Picard, G., and Boissier, O. (2012). Multi-Agent based governance model for Machine-to-Machine networks in a smart parking management system. In *Proceedings of IEEE International Conference on Communications, ICC 2012, Ottawa, ON, Canada, June 10-15, 2012, 3rd IEEE International Workshop on SmArt COmmunications in NEtwork Technologies ('ICC'12 WS - SaCoNet-III')*, pages 6468–6472. IEEE Computer Society.
- Persson, C., Picard, G., Ramparany, F., and Boissier, O. (2012a). A JaCaMo-Based Governance of Machine-to-Machine Systems. In Demazeau, Y., Müller, J. P., Rodríguez, J. M. C., and Pérez, J. B., editors, *Advances on Practical Applications of Agents and Multiagent Systems, Proc. of the 10th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 12)*, volume 155 of *Advances in Soft Computing Series*, pages 161–168. Springer.
- Persson, C., Picard, G., Ramparany, F., and Boissier, O. (2011). A Multi-Agent Organization for the Governance of Machine-to-Machine Systems. In *European Workshop on Multi-agent Systems (EUMAS'11)*.
- Persson, C., Picard, G., Ramparany, F., and Boissier, O. (2011). Organisation multi-agent pour la gouvernance de systèmes Machine-to-Machine. In *19es Journées francophones des systèmes multi-agents (JFSMA'11)*, pages 11–20. Cépaduès.
- Persson, C., Picard, G., and Ramparany, F. (2011a). A Multi-Agent Organization for the Governance of Machine-To-Machine Systems. In *IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'11)*, pages 421–424. IEEE Computer Society.

Bibliography

Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., and Cayirci, E. (2002). Wireless sensor networks: a survey. *Computer networks*, 38(4):393–422. (Cited pages: 20, 23, 26, 33, 34, 35 and 88.)

Alberola, J. M., Búrdalo, L., Julián, V., Terrasa, A., and García-Fornes, A. (2012). Dynamic Monitoring for Adapting Agent Organizations. In Botti Navarro et al. (2012), pages 121–134. (Cited pages: xi, 80, 83, 84, 91 and 92.)

Aldewereld, H., Dignum, V., and Picard, G., editors (2009). *Engineering Societies in the Agents World X, 10th International Workshop, (ESAW X). Proceedings*, volume 5881 of *Lecture Notes in Computer Science*, Utrecht, The Netherlands. Springer. (Cited pages: 228, 229, 233 and 234.)

Aldewereld, H., Padget, J., Vasconcelos, W., Vázquez-Salceda, J., Sergeant, P., and Staikopoulos, A. (2010). Adaptable, Organization-Aware, Service-Oriented Computing. *IEEE Intelligent Systems*, 25(4):26–35. (Cited pages: 58 and 91.)

Andrzejak, A., Graupner, S., Kotov, V., and Trinks, H. (2002). Algorithms for self-organization and adaptive service placement in dynamic distributed systems. *HP Labs*. (Cited pages: 42, 43, 44, 45, 46 and 90.)

Anthony, R. and Mccann, J. (2007). Emergent self-organisation of wireless sensor networks. In *Second Intl. Workshop on Engineering Emergence in Decentralised Autonomic Systems*, pages 2–11. (Cited pages: 26 and 66.)

Arregui, D., Pacull, F., and Willamowski, J. (2001). Addressing Scalability Issues using the CLF Middleware. In *Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International*, pages 93–104. IEEE. (Cited pages: 48 and 90.)

Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer Networks*, 54(15):2787–2805. (Cited pages: 19 and 20.)

Baldoni, M., Baroglio, C., Bergenti, F., Boccalatte, A., and Marengo, E. (2010). MERCURIO: An Interaction-oriented Framework for Designing, Verifying and Programming Multi-Agent Systems. In Boissier, O., El Fallah-Seghrouchni, A., Hassas, S., and Maudet, N., editors, *Proceedings of The Multi-Agent Logics, Languages, and Organisations Federated Workshops (MALLOW 2010)*, volume 627 of *CEUR Workshop Proceedings*, pages 66–83, Lyon, France. CEUR-WS.org. (Cited page: 65.)

Barkai, J. (2008). How to design your business (and products) for Machine to Machine communication. Connected World Conference, Keynote Speech. http://www.connectedworldmag.com/conference/index.php?q=2008-Presentations. (Cited pages: 22 and 34.)

Barreto, C., Kavantzas, N., Fletcher, T., Burdett, D., Ritzinger, G., and Lafon, Y. (2005). Web Services Choreography Description Language Version 1.0. Candidate Recommendation, W3C. http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/. (Cited page: 17.)

Barthel, D., Madhusudan, G., and Dejean, N. (2010). Application characteristics – An applicative framework for the research work conducted in ARESA2. Deliverable Report D1.1, ARESA2 Project, ANR 2009 Verso 017-01. http://aresa-project.insa-lyon.fr/. (Cited pages: xiii, 21, 24, 35, 101 and 119.)

Bauer, B., Müller, J., and Odell, J. (2001). Agent UML: A Formalism for Specifying Multiagent Software Systems. In Ciancarini, Paolo and Wooldridge, Michael, editor, *Agent-Oriented Software Engineering*, volume 1957 of *Lecture Notes in Computer Science*, pages 109–120. Springer Berlin / Heidelberg. (Cited pages: 62, 69 and 70.)

Baugé, T. and Bernat, J. (2008). Components for End to End Networking, Management and Security and Accounting. Deliverable report WP3 / D3.1, SENSEI. (Cited pages: 27 and 88.)

Bellifemine, F., Caire, G., Poggi, A., and Rimassa, G. (2008). JADE: A software framework for developing multi-agent applications. Lessons learned. *Information and Software Technology*, 50(1):10–21. (Cited pages: 65 and 92.)

Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., and Santi, A. (2011). Multi-Agent Oriented Programming with JaCaMo. *Science of Computer Programming*. (Cited pages: 62, 63 and 91.)

Bondi, A. B. (2000). Characteristics of scalability and their impact on performance. In *Proceedings of the 2nd international workshop on Software and performance*, pages 195–203. ACM. (Cited pages: 33, 34 and 35.)

Bondi, A. B. (2009). The software architect as the guardian of system performance and scalability. In *Proceedings of the 2009 ICSE Workshop on Leadership and Management in Software Architecture*, pages 28–31. IEEE Computer Society. (Cited pages: 34 and 37.)

Bora, S. and Dikenelli, O. (2009). Replication Based on Role Concept for Multi-Agent Systems. In Aldewereld et al. (2009), pages 165–180. (Cited pages: 78, 79, 82, 84 and 91.)

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2008). *Programming multi-agent systems in AgentSpeak using Jason*, volume 8. Wiley-Interscience. (Cited pages: 63, 181 and 182.)

Botti Navarro, V. J., Ricci, A., García Fornes, A. M., Weyns, D., Such, J. M., Alberola, J. M., and Pechoucek, M., editors (2012). *Proceedings of The Third International Workshop on Iinfraestructures and tools for multiagent systems : ITMAS 2012*. (Cited pages: 227 and 236.)

Boukhelef, Djelloul Kitagawa, H. (2010). Efficient Load Balancing Techniques for Self-organizing Content Addressable Networks. *Journal of Networks*, 5(3):321–334. (Cited page: 37.)

Bratman, M. E., Israel, D. J., and Pollack, M. E. (1998). Plans and resource-bounded practical reasoning. *Computational intelligence*, 4(3):349–355. (Cited page: 62.)

Brazier, F., Dignum, F., Dignum, V., Huhns, M., Lessner, T., Padget, J., Quillinan, T., and Singh, M. (2010). Governance of Services: A Natural Function for Agents. In *Proceedings of the 8th AAMAS Workshop on Service-Oriented Computing: Agents, Semantics, and Engineering (SOCASE)*, pages 8–22. (Cited pages: 62, 63, 71, 75, 76 and 91.)

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J. (2004). TROPOS: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236. (Cited pages: 66, 68, 75, 76 and 91.)

Brown, W. A. and Cantor, M. (2006). SOA governance: how to oversee successful implementation through proven best practives and methods. White Paper, IBM Rational Center. (Cited pages: xiii, 14, 15, 16, 17, 18 and 89.)

Busetta, P., Rönnquist, R., Hodgson, A., and Lucas, A. (2000). JACK Intelligent Agents-Components for Intelligent Agents in Java. *AgentLink News Letter*, 2(1):2–5. (Cited pages: 63 and 69.)

Capera, D., Georg'e, J.-P., Gleizes, M.-P., and Glize, P. (2003). The AMAS theory for complex problem solving based on self-organizing cooperative agents. In Fredriksson, M., Gustavsson, R., Omicini, A., and Ricci, A., editors, *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on*, pages 383–388. (Cited pages: 46, 66, 70, 80, 84 and 90.)

Carr, H., Pitt, J., Kleerekoper, A., and Blancke, D. (2009). Energy Trade-Offs in Resource-Constrained Multi-Agent Systems. In Aldewereld et al. (2009), pages 113–115. (Cited pages: 46, 66 and 90.)

Carron, T., Proton, H., and Boissier, O. (1999). A temporal agent communication language for dynamic multi-agents systems. In Garijo, F. and Boman, M., editors, *Multi-Agent System Engineering – Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'99)*, pages 115–127. Springer-Verlag : Heidelberg, Germany. (Cited pages: 70 and 76.)

Chan, K. and Sterling, L. (2003). Specifying Roles within Agent-Oriented Software Engineering. In *10th Asia-Pacific Software Engineering Conference (APSEC 2003)*, pages 390–395, Chiang Mai, Thailand. IEEE Computer Society. (Cited pages: 73, 75, 76 and 91.)

Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2001). Web services description language (WSDL) 1.1. Note, W3C. http://www.w3.org/TR/2001/NOTE-wsdl-20010315. (Cited page: 17.)

CTB (2006). Scalability. Canadian Translation Bureau. http://btb.termiumplus.gc.ca/. (Cited page: 29.)

Dastani, M., El Fallah-Seghrouchni, A., Hübner, J., and Leite, J., editors (2011). *Languages, Methodologies, and Development Tools for Multi-Agent Systems - Third International Workshop, LADS 2010, Lyon, France, August 30 - September 1, 2010, Revised Selected Papers*, volume 6822 of *Lecture Notes in Computer Science*. Springer. (Cited pages: 234 and 235.)

de Oliveira Luna, A. J. H., Costa, C. P., de Moura, H. P., Novaes, M. A., and do Nascimento, C. A. (2010). Agile Governance in Information and Communication Technologies: shifting paradigms. *Journal of Information Systems and Technology Management (JISTEM)*, 7(2):311–334. 10.4301/S1807-17752010000200004. (Cited pages: 12 and 13.)

Demazeau, Y. (1995). From Interactions To Collective Behaviour In Agent-Based Systems. In *Proceedings of the 1st. European Conference on Cognitive Science. Saint-Malo*, pages 117–132. (Cited pages: 62, 73, 74 and 91.)

Di Marzo Serugendo, G., Gleizes, M.-P., and Karageorgos, A. (2006). Self-Organisation and Emergence in MAS: An Overview. *Informatica*, 30:45–54. (Cited pages: 65 and 66.)

Dignum, V. (2004). *A Model for Organizational Interaction: Based on Agents, Founded in Logic*. Phd dissertation, Universiteit Utrecht. SIKS dissertation series 2004-1. (Cited pages: 58, 66, 67 and 91.)

Dirks, S. and Keeling, M. (2009). A vision of smarter cities: How cities can lead the way into a prosperous and sustainable future. Technical report, IBM Institute for Business Value study. (Cited page: 19.)

Dragone, M., Jordan, H., Lillis, D., and Collier, R. W. (2011). Separation of Concerns in Hybrid Component and Agent Systems. *International Journal of Communication Networks and Distributed Systems*, 6(2):176–201. (Cited pages: 63, 74, 75, 76 and 92.)

Dressler, F. (2006). Self-Organization in Ad Hoc Networks: Overview and Classification. Technical report, University of Erlangen, Dept. of Computer Science. (Cited page: 45.)

Duboc, L., Rosenblum, D., and Wicks, T. (2007). A framework for characterization and analysis of software system scalability. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC-FSE '07, pages 375–384, New York, NY, USA. ACM. (Cited pages: 31 and 32.)

Erdene-Ochir, O., Minier, M., Valois, F., and Kountouris, A. (2011). Enhancing Resiliency Against Routing Layer Attacks in Wireless Sensor Networks: Gradient-based Routing in Focus. *International Journal on Advances in Networks and Services*. (Cited pages: 26, 42, 43, 88 and 90.)

Esteva, M., Rodriguez-Aguilar, J.-A., Sierra, C., Garcia, P., and Arcos, J. L. (2001). On the Formal Specification of Electronic Institutions. In Dignum, F. and Sierra, C., editors, *Proceedings of the Agent Mediated Electronic Commerce, The European AgentLink Perspective*, volume 1991 of *Lecture Notes in Computer Science*, pages 126–147. Springer Berlin Heidelberg. (Cited pages: 66 and 67.)

Estrin, D., Govindan, R., and Heidemann, J. (2000). Embedding the Internet: introduction. *Commun. ACM*, 43(5):38–41. (Cited pages: 19, 33, 34 and 35.)

ETSI (2011). Technical Specification 102 690 V 0.14.2, Machine-to-Machine communications – Functional architecture. Technical Specification <DRAFT>, European Telecommunications Standards Institute (ETSI). Technical Committee Machine-to-Machine Communications (M2M). (Cited pages: xiii, 22, 28, 29, 89, 100, 113, 114 and 241.)

Fayad, M. E., Laitinen, M., and Ward, R. P. (2000). Thinking objectively: Software Engineering in the Small. *Communications of the ACM*, 43(3):115–118. (Cited page: 37.)

Ferber, J., Gutknecht, O., and Michel, F. (2004). From Agents to Organizations: an Organizational View of Multi-Agent Systems. *Agent-Oriented Software Engineering IV*, pages 443–459. (Cited pages: 66 and 73.)

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University Of California, Irvine. (Cited page: 16.)

Finin, T., Fritzson, R., McKay, D., and McEntire, R. (1994). KQML as an agent communication language. In Adam, N. R., Bhargava, B. K., and Yesha, Y., editors, *Proceedings of the third International Conference on Information and Knowledge Management*, pages 456–463. ACM. (Cited page: 65.)

FIPA (2000). FIPA Interaction Protocol Library Specification. Technical report, Foundation for Intelligent Physical Agents. Technical Report Nb. XC00025E. (Cited pages: 70, 76 and 186.)

Firesmith, D. (2010). Profiling Systems Using the Defining Characteristics of Systems of Systems (SoS). Technical report, Software Engineer Institute, Pittsburgh, Pennsylvania. (Cited pages: 32, 33, 34, 35 and 36.)

Foschini, L., Taleb, T., Corradi, A., and Bottazzi, D. (2011). M2M-Based Metropolitan Platform for IMS-Enabled Road Traffic Management in IoT. *Communication Magazine, IEEE*, 49(11):50–57. (Cited pages: 19, 21 and 88.)

Glaser, N. and Morignot, P. (1997). The reorganization of societies of autonomous agents. In Boman, Magnus and Van de Velde, Walter, editor, *Multi-Agent Rationality*, Lecture Notes in Computer Science, pages 98–111. Springer Berlin / Heidelberg. (Cited pages: 65, 77, 78, 79, 83 and 91.)

Gold, R., Gluhak, A., Bui, N., Waller, A., Sorge, C., Montagut, F., VStirbu, Karvonen, H., Tsiatsis, V., Pennington, S., Shelby, Z., Vercher, J., Johansson, M., JBohli, Bauge, T., Esfandiyari, S., Haller, S., Kovacs, E., Egan, R., Carrez, F., and Presser, M. (2008). State of The Art - Sensor Frameworks and Future Internet. Deliverable report WP3 / D3.1, SENSEI. (Cited pages: 21, 26, 33 and 34.)

Greaves, M., Holmback, H., and Bradshaw, J. (2000). What is a conversation policy? In Dignum, F. and Greaves, M., editors, *Issues in Agent Communication*, volume 1916 of *Lecture Notes in Computer Science*, pages 118–131. Springer Berlin Heidelberg. (Cited pages: 71 and 76.)

Guldentops, E., De Haes, S., Hardy, G., Ormsby, J., Ramos, D. F., Singleton, J., and Williams, P. A. (2003). *Board Briefing on IT Governance*. IT Governance Institute, second edition. (Cited pages: 10 and 11.)

Hayes, T., Blecher, M., Chatterji, S., Jogani, A., Lainhart IV, J. W., Lomparte, R., Schirmbrand, M., and Saull, R., editors (2007). *COBIT 4.1 Excerpt*. IT Governance Institute. (Cited pages: xiii, 10, 14 and 89.)

Heinzelman, W. R., Chandrakasan, A., and Balakrishnan, H. (2000). Energy-efficient communication protocol for wireless microsensor networks. In *System Sciences, Proceedings of the 33rd Annual Hawaii International Conference on*, page 10. IEEE. (Cited page: 45.)

Herssens, C., Faulkner, S., and Jureta, I. (2010). Normative Management of Web Service Level Agreements. In Benatallah, B., Casati, F., Kappel, G., and Rossi, G., editors, *Web Engineering*, volume 6189 of *Lecture Notes in Computer Science*, pages 98–113. Springer Berlin / Heidelberg. (Cited pages: 18, 72, 75, 76 and 91.)

Herstad, A., Nersveen, E., Samset, H., Storsveen, A., Svaet, S., and Husa, K. E. (2009). Connected objects: Building a service platform for M2M. In *Intelligence in Next Generation Networks, 2009. ICIN 2009. 13th International Conference on*, pages 302–304. IEEE. (Cited pages: 26 and 29.)

Hewitt, C. (1977). Viewing control structures as patterns of passing messages. *Artificial intelligence*, 8(3):323–364. (Cited pages: 70 and 76.)

Hill, M. D. (1990). What is scalability? *ACM SIGARCH Computer Architecture News*, 18(4):18–21. (Cited pages: 29 and 32.)

Horling, B. and Lesser, V. (2004). A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(04):281–316. (Cited pages: 65, 66 and 73.)

Hübner, J. F. and Boissier, O. (2002). A Model for the Structural, Functional, and Deontic Specification of Organizations in Multiagent Systems. In Bittencourt, G. and Ramalho, G., editors, *Advances in Artificial Intelligence*, volume 2507 of *Lecture Notes in Computer Science*, pages 439–448. Springer Berlin / Heidelberg. (Cited pages: 66, 67, 73, 75, 76, 78 and 91.)

Hübner, J. F., Boissier, O., Kitio, R., and Ricci, A. (2010). Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems*, 20(3):1387–2532. (Cited pages: 74 and 83.)

Hübner, J. F. and Sichman, J. S. and Boissier, O. (2004). Using the MOISE+ for a cooperative framework of MAS reorganisation. In *17th Brazilian Symposium on Artificial Intelligence (SBIA'04)*, pages 506–515. Springer. (Cited pages: 78, 79, 81, 83, 91 and 128.)

Huebscher, M. C. and McCann, J. A. (2008). A survey of Autonomic Computing—Degrees, Models, and Applications. *ACM Computing Survey*, 40(3):1–28. (Cited pages: 52 and 56.)

IBM (2006). An architectural blueprint for autonomic computing. White Paper, IBM. Fourth edition. (Cited pages: 18, 47, 52 and 90.)

Iwanicki, K. and van Steen, M. (2010). Gossip-Based Self-Management of a Recursive Area Hierarchy for Large Wireless SensorNets. *Parallel and Distributed Systems, IEEE Transactions on*, 21(4):562–576. (Cited pages: 26, 42 and 45.)

Jennings, N. R. (2001). An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41. (Cited pages: xi, 46, 62, 63, 66, 73, 74 and 90.)

JO (2003). Extensibilité. Journal Officiel (French). (Cited page: 29.)

Kai, K., Xuguang, W., and Jie, Z. (2009). The Research of the Dimensions of Governance Performance in Network Organizations. In *International Conference on Information Management, Innovation Management and Industrial Engineering*, pages 410–413. IEEE Computer Society. (Cited page: 13.)

Kessis, M., Roncancio, C., and Lefebvre, A. (2009). DASIMA: A flexible Management Middleware in Multi-Scale Contexts. In *Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on*, pages 1390–1396. IEEE. (Cited pages: 36, 37, 49 and 90.)

Kota, R., Gibbins, N., and Jennings, N. R. (2012). Decentralised Approaches for Self-Adaptation in Agent Organisations. *ACM Transactions on Autonomous and Adaptive Systems*, 7(1):1–28. (Cited pages: 42, 46, 78, 80, 81, 90, 91 and 92.)

Kousaridas, A., Boite, J., Conan, V., Raptis, T., and Alonistioti, N. (2010). *An experimental path towards Self-Management for Future Internet Environments*, pages 95–104. IOS Press. (Cited pages: 57 and 90.)

Kubera, Y., Mathieu, P., and Picault, S. (2008). Interaction-Oriented Agent Simulations : From Theory to Implementation. In Ghallab, M., Spyropoulos, C., Fakotakis, N., and Avouris, N., editors, *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)*, pages 383–387. IOS Press. (Cited pages: 65, 66, 71 and 76.)

Lam, J., Vasconcelos, W., Guerin, F., Corsar, D., Chorley, A., Norman, T., Vázquez-Salceda, J., Panagiotidi, S., Confalonieri, R., Gomez, I., et al. (2009). ALIVE: A Framework for Flexible and Adaptive Service Coordination. In Aldewereld et al. (2009), pages 236–239. (Cited page: 88.)

Lampin, Q., Barthel, D., Augé-Blum, I., and Valois, F. (2012). QoS oriented Opportunistic Routing protocol for Wireless Sensor Networks. In *Proceedings of the IFIP Wireless Days Conference 2012, Ireland, November 21-23, 2012*, pages 1–6. IEEE Xplore. (Cited pages: 26, 35 and 88.)

Lesser, V., Decker, K., Wagner, T., Carver, N., Garvey, A., Horling, B., Neiman, D., Podorozhny, R., Prasad, M. N., Raja, A., Vincent, R., Xuan, P., and Zhang, X. (2004). Evolution of the GPGP/TÆMS Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems*, 9(1):87–143. (Cited page: 66.)

Ludwig, H., Keller, A., Dan, A., King, R. P., and Franck, R. (2003). Web Service Level Agreement (WSLA) Language Specification. Technical Report 1.0, IBM T.J. Watson Research Center. (Cited pages: 17 and 89.)

Ma, X. and Luo, W. (2008). The analysis of 6LoWPAN technology. In *Computational Intelligence and Industrial Application, 2008. PACIIA'08. Pacific-Asia Workshop on*, volume 1, pages 963–966. IEEE. (Cited page: 20.)

Madhusudan, G. and Bottaro, A. (2009). City Automation, Technology for city e automation services. White Paper, Orange Labs, Grenoble, France. (Cited pages: 19, 20, 22, 23, 33, 34 and 35.)

Martsola, M., Kiravuo, T., and Lindqvist, J. (2005). Machine to machine communication in cellular networks. In *Mobile Technology, Applications and Systems, 2005 2nd International Conference on*, pages 6–pp. IEEE. (Cited page: 20.)

Maurel, Y. (2010). *CEYLAN : Un canevas pour la création de gestionnaires autonomiques extensibles et dynamiques*. PhD thesis, Université de Grenoble. French. (Cited pages: 18, 55, 56 and 90.)

Melliar-Smith, P., Moser, L., and Narasimhan, P. (1997). Separation of Concerns: Functionality vs. Quality of Service. In *Object-Oriented Real-Time Dependable Systems, 1997. Proceedings., Third International Workshop on*, pages 272–274. IEEE. (Cited page: 62.)

Michael, M., Moreira, J. E., Shiloach, D., and Wisniewski, R. W. (2007). Scale-up x scale-out: A case study using nutch/lucene. In *3rd International Workshop on System Management Techniques, Processes, and Services (SMTPS). Held in conjunction with the 21stParallel and Distributed Processing Symposium (IPDPS 2007)*, pages 1–8. IEEE. (Cited pages: 33, 36 and 37.)

Miller, R. and Shanahan, M. (1999). The Event Calculus in Classical Logic - Alternative Axiomatisations. *Electronic Transactions on Artificial Intelligence*, Vol. 3, Section A:77–105. http://www.ep.liu.se/ej/etai/1999/016/. (Cited page: 72.)

Minotti, M., Ricci, A., and Santi, A. (2011). Exploiting Agent-Oriented Programming for Developing Future Internet Applications Based on the Web: The JaCa-Web Framework. In Dastani et al. (2011), pages 76–94. (Cited page: 74.)

Mitra, N. (2003). SOAP version 1.2 part 0: Primer. first edition of a recommendation, W3C. http://www.w3.org/TR/2003/REC-soap12-part0-20030624/. (Cited page: 17.)

Morandini, M., Migeon, F., Gleizes, M.-P., Maurel, C., Penserini, L., and Perini, A. (2009). A Goal-Oriented Approach for Modelling Self-organising MAS. In Aldewereld et al. (2009), pages 33–48. (Cited pages: 79, 80, 82, 84 and 91.)

Moritsch, H. (2008). Scalability evaluation methodology and schedule. Presentation, TripCom. (Cited page: 30.)

Morrish, J. (2012). Mobile Operator Strategies for Success in M2M: The Role of Cloud Platforms. Technical report, Machina Research. (Cited pages: 21, 22 and 24.)

Moss, J. (2011). Machine-to-Machine: Operator strategy and best practice. Technical Report 6, Informa UK, http://www.informatm.com. (Cited page: 35.)

Neuman, B. C. (1994). Scale in Distributed Systems. In *Readings in Distributed Computing Systems*, pages 463–489. IEEE. (Cited pages: 33, 34, 35, 36 and 37.)

Northrop, L., Feiler, P., Gabriel, R. P., Goodenough, J., Linger, R., Longstaff, T., Kazman, R., Klein, M., Schmidt, D., Sullivan, K., and Wallnau, K. (2006). Ultra-Large-Scale Systems – The Software Challenge of the Future. Technical report, Software Engineering Institute, Carnegie Mellon. (Cited pages: 35, 36 and 37.)

OASIS (2007). Web Services Business Process Execution Language. OASIS Standard, Organization for the Advancement of Structured Information Standards (OASIS). WSBPEL Technical Committee. (Cited page: 17.)

OECD (2012). Machine-to-Machine Communications: Connecting Billions of Devices. *OECD Digital Economy Papers*, No. 192, OECD Publishing.
http://dx.doi.org/10.1787/5k9gsh2gp043-en. (Cited pages: 25 and 29.)

Omicini, A., Ricci, A., and Viroli, M. (2008). Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456. (Cited pages: xi, 64, 74, 75 and 92.)

Oren, N., Preece, A., and Norman, T. J. (2007). Argument Based Contract Enforcement. In Bramer, M., Coenen, F., and Tuson, A., editors, *Research and Development in Intelligent Systems XXIII*, pages 145–158. Springer London. (Cited pages: 78, 79 and 83.)

Overeinder, B., Wijngaards, N., van Steen, M., and Brazier, F. (2002). Multi-agent support for Internet-scale Grid management. In *Proceedings of Artificial Intelligence and Simulation of Behavior (AISB)*, volume 2, pages 18–22. (Cited pages: 56, 66 and 90.)

Oyenan, W. H., Deloach, S. A., and Singh, G. (2010). An Organizational Design for Adaptive Sensor Networks. In *Proceedings of the 2010 IEE/WIC/ACM Intl Conf on Intelligent Agent Technology*, volume 2, pages 239–242. (Cited pages: 57 and 90.)

Picard, G. and Gleizes, M.-P. (2004). The ADELFE Methodology – Designing Adaptive Cooperative Multi-Agent Systems. In *Methodologies and Software Engineering for Agent Systems*, volume 11 of *Multiagent Systems, Artificial Societies, And Simulated Organizations*, chapter 8, pages 157–176. Kluwer Publishing. (Cited pages: 43, 69, 75, 76, 91 and 92.)

Picard, Gauthier Boissier, O., Gleizes, M.-P., and Hübner, J. F. (2009). Reorganisation and Self-organisation in Multi-Agent Systems. In *International Workshop on Organizational Modeling (OrgMod'09)*, pages 66–80. (Cited pages: xiii, 66, 79 and 91.)

Pitt, J., Venkataram, P., and Mamdani, A. (2006). QoS Management in MANETs Using Norm-Governed Agent Societies. In Dikenelli, O., Gleizes, M.-P., and Ricci, A., editors, *Engineering Societies in the Agents World VI*, volume 3963 of *Lecture Notes in Computer Science*, pages 221–240. Springer Berlin / Heidelberg. (Cited pages: 72, 74, 75, 76 and 91.)

Piunti, M., Santi, A., and Ricci, A. (2009). Programming SOA/WS Systems with Cognitive Agents and Artifact-Based Environments. In Baldoni, M., Baroglio, C., Bentahar†, J., and Mascardi, V., editors, *Proceedings of the Multi-Agent Logics, Languages, and Organisations, Federated Workshops, MALLOW'009, Agent, Web Services and Ontologies, Integrated Methodologies (MALLOW-AWESOME'009) workshop, Turin, Italy*. CEUR-WS.org. (Cited page: 74.)

Populaire, P., Demazeau, Y., Boissier, O., and Sichman, J. S. (1993). Description et implémentation de protocoles de communication en univers multi-agents. In *Premières des Journées Francophones Intelligence Artificielle Distribuée et Systèmes Multi-Agents (JFIADSMA'93)*, Toulouse, France. (Cited pages: 70 and 76.)

Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In Van de Velde, W. and W. Perram, J., editors, *Proceedings of the 7th European workshop on Modelling Autonomous Agents in a Multi-Agent World: Agents Breaking Away (MAAMAW '96)*, pages 42–55. Springer-Verlag New York, Inc. (Cited pages: 63 and 182.)

Römer, K., Kasten, O., and Mattern, F. (2002). Middleware challenges for wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Revue*, 6(4):59–61. (Cited page: 34.)

Santi, A., Guidi, M., and Ricci, A. (2011). JaCa-Android: An Agent-Based Platform for Building Smart Mobile Applications. In Dastani et al. (2011). (Cited page: 74.)

Santucci, G. (2008). Policy and Technological Drivers in the Internet of Things. Internet of Things 2008 Conference, Zurich, Switzerland, Official Opening Speech. http://www.internet-of-things-research.eu/documents.htm. (Cited page: 22.)

Schoonderwoerd, R., Holland, O., Bruten, J., and Rothkrantz, L. (1997). Ant-based Load Balancing in Telecommunications Networks. *Adaptive Behavior*, 5(2):169–207. (Cited pages: 42, 44 and 90.)

Shelby, Z. (2010). Embedded Web Services. *Wireless Communications, IEEE*, 17(6):52–57. (Cited page: 26.)

Sorici, A., Picard, G., Boissier, O., Santi, A., and Hübner, J. F. (2012). Multi-Agent Oriented Reorganisation within the JaCaMo infrastructure. In Botti Navarro et al. (2012), pages 135–148. (Cited pages: xi, 81, 82, 84, 85, 91, 92 and 128.)

Souza, V. E. S., Lapouchnian, A., Robinson, W. N., and Mylopoulos, J. (2011). Awareness Requirements for Adaptive Systems. In Giese, H. and Cheng, B. H., editors, *Proceeding of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '11, pages 60–69, New York, NY, USA. ACM. (Cited page: 18.)

Spiess, P., Karnouskos, S., Guinard, D., Savio, D., Baecker, O., Souza, L. M. S., and Trifa, V. (2009). SOA-based Integration of the Internet of Things in Enterprise Services. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 968–975. IEEE. (Cited pages: 21, 28, 29 and 89.)

Sycara, K., Norman, T., Giampapa, J., Kollingbaum, M., Burnett, C., Masato, D., McCallum, M., and Strub, M. (2009). Agent Support for Policy-Driven Collaborative Mission Planning. *The Computer Journal*, 53(5):528–540. (Cited pages: 72, 74, 75, 76, 91 and 92.)

Tambe, M. (1997). Towards Flexible Teamwork. *Journal of Artificial Intelligence Reseearch*, 7:83–124. (Cited page: 66.)

Toomey, M. (2009). *Waltzing with the Elephant: A comprehensive guide to directing and controlling information technology.* Infonomics Pty Ltd., first edition. Promotional extract (62 pages). (Cited page: 10.)

Ückelmann, D., Harrison, M., and Michahelles, F., editors (2011). *Architecting the Internet of Things*. Springer, 1st edition. (Cited pages: 21, 29, 35 and 88.)

Ückelmann, D., Isenberg, M., Teucke, M., Halfar, H., and Scholz-Reiter, B. (2010). Autonomous Control and the Internet of Things: Increasing Robustness, Scalability and Agility in Logistic Networks. *Unique Radio Innovation for the 21st Century*, pages 163–181. (Cited page: 88.)

Van Steen, M., Van der Zijden, S., and Sips, H. (1998). Software engineering for the scalable distributed applications. In *Proceedings of the 22nd Annual International Computer Software and Applications Conference (COMPSAC'98)*, pages 285–292. IEEE. (Cited pages: 30 and 32.)

Vitvar, T., Vinoski, S., and Pautasso, C. (2012). Programmatic Interfaces for Web Applications. *IEEE Internet Computing*, 16(4):11–14. (Cited pages: 16 and 17.)

Wefer, Gerold, editor (2009). *Cyberinfrastructure for the US Ocean Observatories Initiative: Enabling interactive observation in the ocean*. IEEE Xplore. (Cited pages: 28 and 89.)

Weinstock, C. B. and Goodenough, J. B. (2006). On System Scalability. Technical report, Software Engineer Institute. (Cited pages: 30, 32, 34, 35, 36 and 37.)

Weyns, D., Malek, S., and Andersson, J. (2010). On decentralized self-adaptation: lessons from the trenches and challenges for the future. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '10, pages 84–93. ACM. (Cited pages: xi, 54, 55, 56 and 90.)

Weyns, D., Omicini, A., and Odell, J. (2007). Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agents Systems*, 14(1):5–30. (Cited pages: xi, 63, 64 and 92.)

Weyuker, E. J. and Avritzer, A. (2002). A metric to predict software scalability. In *Proceedings of the 8th IEEE Symposium on Software Metrics (METRICS '02)*, pages 152–158. IEEE. (Cited pages: 33, 34 and 35.)

Wijngaards, N., Overeinder, B., van Steen, M., and Brazier, F. (2002). Supporting Internet-Scale Multi-Agent Systems. *Data and Knowledge Engineering*. (Cited page: 56.)

Wilson, D., Birks, F., Gold, R., Holler, J., Darianian, M., Van Gurp, J., Eurich, M., Petcu, A., Haller, S., Tierno, A., Egan, R., Lavoisy, O., Fabrice Forest, Shelby, Z., Meratnia, N., and Strohbach, M. (2009). SENSEI Scenario Portfolio, User and Context Requirements. Deliverable report WP1 / D1.1, SENSEI. (Cited pages: 33, 34 and 35.)

Yu, E. S. (1997). Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In Heitmeyer, C., editor, *Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on*, pages 226–235, Annapolis, Maryland, USA. IEEE Computer Society, ACM SIGSOFT, IEEE Computer Society Press. (Cited pages: 18, 67, 75 and 76.)

Appendix

Appendix A

ETSI M2M Service Capabilities

This appendix summarizes the Machine-to-Machine (M2M) functional specifications recommended by the European Telecommunications Standards Institute Technical Committee on M2M (ETSI TC M2M), and described in ETSI (2011). It is composed 11 capabilities which are decomposed into functionalities. Although these capabilities are defined for the network, the gateway and the device domains, we focus on the network layer, the Network Service Capability Layer (NSCL), as it is the one our approach is based on. For each capability, we describe the list of functionality it provides.

A.1 Network Application Enablement (NAE) capability

The NAE Capability in the NSCL is the single contact point to M2M applications. It provides the following functionalities:

- Exposes functionalities implemented in NSCL via a single reference point.
- Allows M2M applications to register to the NSCL.
- Performs routing between NAs and capabilities in the NSCL. Routing is defined as the mechanism by which a specific request is sent to a particular capability or an instance of that capability when e.g. load balancing is implemented. It also allows routing towards different capabilities.
- Generates charging records pertaining to the use of capabilities.

A.2 Network Generic Communication (NGC) capability

The NGC capability in the NSCL is the single point of contact for communication with Gateway Service Capability Layer (GSCL) and Network Service Capability Layer (DSCL). It provides the following functionalities:

- Provides transport session establishment and teardown along with security key negotiation. If such a transport session establishment is performed in a secure way, then the security key negotiation uses keys as provided by NSEC.
- Provides encryption/integrity protection on data exchanged with the M2M Devices/Gateways. Key material for encryption and integrity protection is derived upon secure session establishment.
- Ensures secure delivery of application data from/to M2M Gateways or M2M Devices and the NSCL when security is required.

- Reports transmission errors.
- Is optionally able to inspect traffic generated by a particular M2M Device or M2M Gateway and verify it is matching a given traffic pattern.

A.3 Network Reachability, Addressing and Repository (NRAR) capability

The NRAR Capability in the NSCL manage groups of devices, routes and subscription to these groups. It provides the following functionalities:

- Provides a mapping between the name of an M2M Device or an M2M Gateway or a group of M2M Devices/M2M Gateways and a set of information:
  - Set of routable network addresses of the M2M Device or M2M Gateway.
  - Reachability status of an M2M Device or M2M Gateway.
  - Scheduling information pertaining to reachability of the M2M Device or M2M Gateway, if available.
- Manages subscriptions and notifications pertaining to events.
- Allows to create, delete and list a group of M2M Devices or M2M Gateways.
- Store NA, DSCL, GSCL related registration information.
- Store application (NA, DA, GA) and SCL data and make it available, on request or based on subscriptions, subject to access rights and permissions.
- Informs NSCL about reachability and scheduling information pertaining to reachability of the M2M Gateway.

A.4 Network Communication Selection (NCS) capability

The NCS Capability in the NSCL manages and selects alternative routes to the M2M devices. It provides the following functionalities:

- Provides network selection, based on policies, when the M2M Device or M2M Gateway can be reached through several networks or several bearers.
- Provides alternative Network or Communication Service selection after a communication failure using a first selected Network or Communication Service.

A.5 Network Remote Entity Management (NREM) capability

The NREM Capability in the NSCL manages the M2M devices' configuration, performance and failures. It provides the following functionalities:

- Provides Configuration Management (CM) functions. CM is the means to provision a set of Management Objects in an M2M Device, an M2M Gateway, a set of M2M Devices or a set of M2M Gateways. Examples of CM include: the Activation of Fault Management (FM) and Performance Management (PM) collection.
- Collects and stores Performance Management (PM) data, e.g. radio interference management data. Provides upon request or other policies the related PM data to NAs and/or M2M Management Functions.
- Collects and stores Fault Management (FM) data. Provides upon request or other policies the related FM data to NAs and/or M2M Management Functions.
- Triggers connection establishment.
- Performs software and firmware upgrade of M2M Device or M2M Gateways
  - The NREM supports independent management of M2M software and M2M firmware for different part of an M2M Device or M2M Gateway by different management authorities. The management authorities supported in an M2M Device or M2M Gateway are:
    - One M2M Service Provider –for management of SCL related M2M software and M2M firmware.
    - One or more Application Providers (including the manufacturer) – for management of M2M software and M2M firmware that is controlling application specific hardware modules in the M2M Device/M2M Gateway.
  - If M2M Device or M2M Gateway integrity validation is supported, the NREM supports NSEC-triggered post-validation actions including remediation, roll-back or update of software and firmware of the M2M Device/M2M Gateways.
  - Information including integrity-protected trusted reference values, policy and configuration related information (i.e. identification of executables for integrity validation) can optionally be included in the post-validation entity-management. procedures,
  - The NREM can optionally support Authentication and Authorization of a management authority as well as integrity checking of M2M software/M2M firmware from these management authorities.
  - The NREM is responsible for avoiding conflicts among multiple authorities.
- Supports several management protocols on the reference point for managing different M2M Devices and M2M Gateways.
- Supports a common mechanism to indicate the management protocol to be used by NREM.
- Supports the following management functions at different layers of remote entities:

  - M2M service management: configuration management for the M2M Service Capabilities in the M2M Device/M2M Gateway.
  - M2M Area Network management: configuration management for the M2M Area Networks.
  - M2M application lifecycle management: installing, removing and upgrading applications in an M2M Device/M2M Gateway.
  - M2M device management : configuration management of the M2M Device/M2M Gateway.

A.6 Network Security Capability (NSEC) capability

The NSEC Capability in the NSCL manages the authentication, key hierarchy management, integrity validation and verification. It provides the following functionalities:

- Supports M2M Service Bootstrap.
- Support key hierarchy realisation for authentication and authorization.
- Performs mutual authentication and key agreement.
- Can verify the integrity validation status reported by the M2M Device/M2M Gateway and trigger appropriate post validation actions.

A.7 Network History and Data Retention (NHDR) capability

The NHDR Capability in the NSCL is an optional capability, i.e. deployed when needed/required by NSCL operator policies. It archives and retains data from the devices. The NHDR Capability provides the following functionalities:

- Archives relevant information pertaining to messages exchanged over the reference point and also internally to the NSCL based on policies.
- Interacts with the other capabilities residing in the NSCL to:
  - determine if and which information requires to be retained;
  - obtain the information to be stored from capability involved.

A.8 Network Transaction Management (NTM) capability

The NTM capability in the NSCL is an optional capability, i.e. deployed when needed/required by operator policies.

For the purpose of this specification, a transaction is an operation that involves several operations (from different entities in the M2M System) and requires atomicity in its execution. In case one single operation is not completed, the overall transaction is considered as unsuccessful and a rollback operation is triggered for operations that were reported to be successful.

The NTM Capabilities provides the following functionalities:

- Propagates the individual operation requests: aggregates the results of the individual operations and commits the transaction when all individual operations have completed successfully.
- Triggers a roll-back if any individual operation fails.

A.9 Network Interworking Proxy (NIP) capability

The NIP Capability in the NSCL is an optional capability, i.e. deployed when needed/required by operator policies. The NIP Capability manages interworking with non-ETSI compliant systems. It provides the following functionalities:

- Provides interworking between non ETSI compliant devices or gateways and the NSCL.
- Depending on the nature of existing non-ETSI compliant M2M deployments, there might be other ways to provide interworking.
- It is recognized that full interworking is not possible in some cases, depending on the characteristics of the M2M Device/Gateway.

A.10 Network Compensation Brokerage (NCB) capability

The NCB Capability in the NSCL is an optional Capability, i.e. deployed when needed/required by operator policies. Compensation involves three roles: a Customer (any application buying a service from a service provider), a Service Provider (any application providing the service) and a Broker implemented by the NCB Capability in the NSCL. Customers buy or acquire services from a Service Provider by applying electronic compensation. A Broker represents a trusted third party who will bill Customers according to their expenditures committed towards Service Providers, and accordingly redeems the Service Providers for such amounts.

The NCB Capability provides the following functionalities:

- Submits compensation tokens (i.e. electronic money) to requesting Customers.
- Verifies the validity of compensation tokens.
- Bills the Customer of compensation tokens for the amount spent.
- Redeems Service Providers for tokens acquired as compensation for services provided to customers.

A.11 Network Telco Operator Exposure (NTOE) capability

The NTOE Capability in the NSCL is an optional Capability, i.e. deployed when needed/required by operator policies. NTOE provides the following functionality:

- Interworking and using of Core Network services exposed by the Network Operator.

APPENDIX B

SensCity Architecture

This appendix describes the Machine-to-Machine (M2M) core platform used in the SensCity project.

The SensCity core platform is deployed as a component-based Web Service server. It is divided into an `Urban Service Platform (USP)` part, to manage applications' subscriptions rights, notifications and billing, and an `Urban Collect & Command Platform (UCCP)` part, to manage devices.

B.1 Urban Collect & Command Platform Components

The `UCCP` enables the message collection from the devices and provides a subscription service. It uses the following components:

The *Device Connection* component enables the connectivity between the `UCCP` with the `Gateways` using a Web Service interface.

The *Device Management and Registry* component manages device provisioning and provides information related to the devices. For each device, it stores applicative data, network data and location. It also comes with two complementary components: the ***Device Registry DB* component** which is responsible for storing the information about the devices into a database, and the ***Device Registry WS* component** which is a Web Service interface allowing to request the *Device Management and Registry* component. It is possible to get device lists depending on different criteria (e.g. location, type). It is also used to subscribe to network changes notifications (e.g. addition/removal of devices, new type of devices).

The *Network Monitoring* component detects network malfunctioning and abnormal behaviors –based on parametrized expected behaviors– and generates alerts accordingly. It comes with a Web Service interface: the ***Network Monitoring WS* component** which is Web Service interface exposing the *Network Monitoring* allowing to configure the devices' expected behavior.

The *Statistics* component traces all the messages sent to/received by the devices and the platform. For each message, it stores information useful for computing statistics about the network or the platform (e.g. timestamps, size, direction). These data are exposed to external systems capable to process them.

The *Store & Forward* component collects the messages sent by the devices, store them until they are forwarded to the `USP`. It also sends messages to the devices, detects duplicated messages and delete them. The stored messages are also regularly archieved in an other database. It comes with two complimentary components: the ***Store & Forward WS* component** which is a Web Interface exposing the *Store & Forward*'s services, and the ***Message DB* component** that stores all the messages from/to the devices in a fast-access database during a contract-define period, then it archives them in an other database.

The *Subscription* component manages USP's subscriptions to the messages sent by the devices and sends the notifications. It comes with a Web Service interface: the ***Subscription WS* component** which is a Web Service Interface exposing the *Subscription*'s services.

B.2 Urban Service Platform Components

The USP provides an interface for the M2M applications to subscribe to devices' messages, to receive such data, and to send commands to the devices. It uses the following components:

The *Accounting* component keeps track of all the messages sent/received by the client applications and store information related to them (e.g. number of messages, priority, size and direction). It comes with six complimentary components. the ***Accounting DB* component**] which manages the accounting information storage, the ***User and Access Rights Management* component** which manages the client `applications` accounts and access rights to the devices, the ***Session Management and Authentification* component** which authenticates the `applications` and manages their sessions, the ***Application Management and Registry* component** which allows to configure the applications subscriptions and to parametrize the data persistence, and two Web Service Interfaces: the ***Management WS* component** which exposes the services of the *User and Access Rights Management* and the *Application Management and Registry* components, and the ***Application WS* component** for the applications to register and connect to the platform.

The *Subscription* component manages the applications' subscriptions and notifications. It comes with complimentary components: the ***Subscription WS* component** which is a Web Service interface exposing the *Subscription*'s services. the ***Message Dispatcher* component** which dispatches the messages received from the UCCP to the proper components, according to the subscriptions, the ***Application DB* component** which temporarily stores the messages destined to the applications, so that they can retrieve them later, and the ***Persistence* component** which stores the data on the platform until they are retrieved by the applications.

The *Device Proxies* component provides a representation of the physical devices and exposes their services. It receives the messages from/to the devices and decode/encode them from/to raw data. It comes with two complementary components: the ***Device Group* component** which enables to define groups of devices (and possibly groups included into other groups). Such a group can be used to process a command or a subscription of several devices at once, and the ***Data Processor & Provider* component** which is an optional generic components to process the messages from the devices and extract the business data.

The *WS Client* component is Web Service interface which provides an access point to the UCCP's services.

The *Log* component keeps track of all the events happening on the platform.

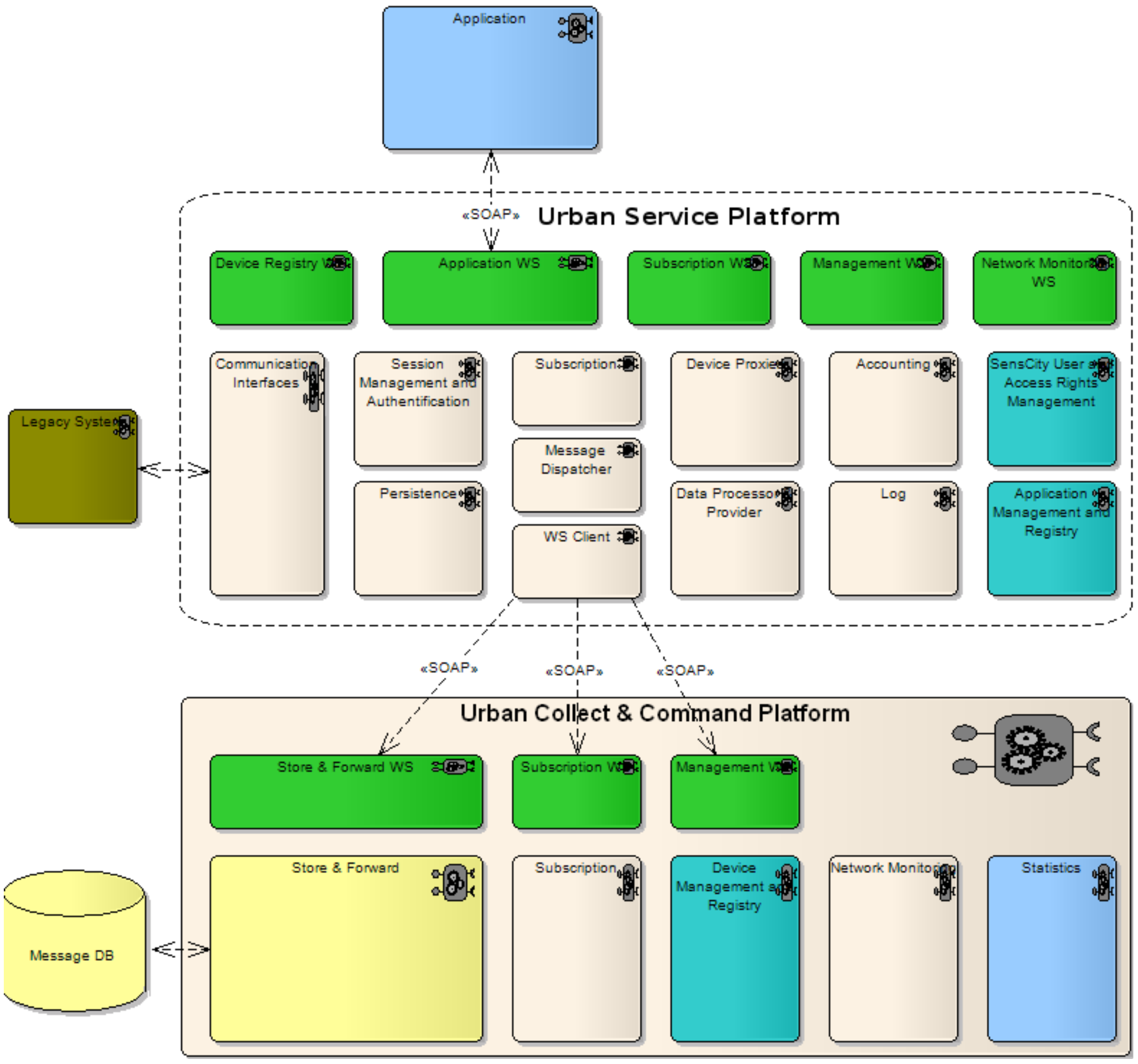


Figure B.1: The *SensCity's core platform* component-based architecture.

Appendix C

Example of a Vertical SLA Organizational Specification (OS)

This appendix provides an Organizational Specification (OS) example representing a vertical Machine-to-Machine (M2M) Service Level Agreement (SLA). This OS is specified using the Moise XML organization description format.

This example is the one corresponding to the `ON_EVENT` SLA profile, used in Chapter 10 during the simulations. It is based on the *Data Collection* template presented in Section 6.3. Using XML entities (Lines 6–16), it is very easy and quick to parametrize from to the SLA established.

C.1 `ON_EVENT` Vertical SLA OS in Moise

Sources/VOS–ONEVENT.xml

```
   <!-- Organisational Specifications for a M2M Vertical SLA -->
 2 <!-- Copyright 2012 Persson -->

 4 <?xml-stylesheet href="http://moise.sourceforge.net/xml/os.xsl" type="text/xsl" ?>

 6 <!-- SLA PARAMETERS -->
   <!DOCTYPE organisational-specification[
 8 <!ENTITY gpID "gpVM2M">     <!-- Group identifier -->
   <!ENTITY R    "ONEVENT">     <!-- QoS parameters -->
10 <!ENTITY F    "ONEVENT">
   <!ENTITY W    "10KB">
12 <!ENTITY Lo   "0.0001">
   <!ENTITY Lat  "3min">
14 <!-- Compensation function -->
   <!ENTITY Fc   "fc(application,device,V,300-V)">
16 ] >

18 <organisational-specification
       id="v100"
20     os-version="0.9"
       xmlns='http://moise.sourceforge.net/os'
22     xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
       xsi:schemaLocation='http://moise.sourceforge.net/os
24                         http://moise.sourceforge.net/xml/os.xsd' >
```

```
<!-- STRUCTURAL SPECIFICATION -->

<structural-specification>
  <role-definitions>
    <role id="m2mRole" />
    <role id="application">
      <extends role="m2mRole" />
    </role>
    <role id="corePlfm">
      <extends role="m2mRole" />
    </role>
    <role id="gateway">
      <extends role="m2mRole" />
    </role>
    <role id="sensor">
      <extends role="m2mRole" />
    </role>
    <role id="actuator">
      <extends role="m2mRole" />
    </role>
    <role id="repeater">
      <extends role="m2mRole" />
    </role>
  </role-definitions>

  <group-specification id="\&gpID;" monitoring-scheme="schCompensation">
    <roles>
      <role id="application" min="1" max="1" />
      <role id="corePlfm" min="1" max="100" />
      <role id="gateway" min="1" max="100" />
      <role id="sensor" min="1" max="100" />
    </roles>

    <links>
      <link from="m2mRole" to="m2mRole" type="communication" scope="intra-group"
            bi-dir="true" />
    </links>
  </group-specification>
</structural-specification>


<!-- FUNCTIONAL SPECIFICATION -->

<functional-specification>
  <scheme id="schDataCollection">
    <goal id="dataCollection" ds="Data Collection scheme for contract \&gpID;">
      <plan operator="parallel">
        <goal id="environmentSensing" ds="sensing\_rate(\&R;)" />
        <goal id="dataTransmission">
          <plan operator="parallel">
```

```
                <goal id="wsnTransmission" ds="freq(\&F;);weight(\&W;);loss(\&Lo;);
                    latency(\&Lat;)" />
                <goal id="nwkTransmission" ds="freq(\&F;);latency(\&Lat;)" />
              </plan>
            </goal>
            <goal id="dataDelivery">
              <plan operator="parallel">
                <goal id="appNotification" ds="freq(\&F;);latency(\&Lat;)" />
                <goal id="appRetrieval" ds="" />
              </plan>
            </goal>
          </plan>
        </goal>

        <mission id="mSense" min="1" >
          <goal id="environmentSensing" />
        </mission>
        <mission id="mNotif" >
          <goal id="wsnTransmission" />
        </mission>
        <mission id="mWSNRepeat" >
          <goal id="wsnTransmission" />
        </mission>
        <mission id="mNwkTransmit" >
          <goal id="nwkTransmission" />
        </mission>
        <mission id="mDelivery">
          <goal id="appNotification" />
          <goal id="appRetrieval" />
        </mission>
      </scheme>

      <scheme id="schCompensation">
        <goal id="gManagedCompensation">
          <plan operator="sequence">
            <goal id="gComputePenalties" ds="Register SLA violations">
              <argument id="V" value=""/>
            </goal>
            <goal id="gBillCompensation" ds="compensation(\&Fc;)"/>
          </plan>
        </goal>

        <mission id="mPenalties">
          <goal id="gComputePenalties"/>
        </mission>
        <mission id="mBill">
          <goal id="gBillCompensation"/>
        </mission>
      </scheme>
    </functional-specification>
```

```
  <!-- NORMATIVE SPECIFICATION -->

  <normative-specification>
    <norm id="\&gpID;nDC0" type="obligation" role="sensor" mission="mSense"   />
    <norm id="\&gpID;nDC1" type="obligation" role="sensor" mission="mNotif"   />
    <norm id="\&gpID;nDC2" type="obligation" role="repeater" mission="mWSNRepeat" />
    <norm id="\&gpID;nDC3" type="obligation" role="gateway" mission="mNwkTransmit" />
    <norm id="\&gpID;nDC4" type="obligation" role="platform"     mission="mDelivery" />
    <norm id="\&gpID;nDC5" type="obligation" role="application" mission="mDelivery" />

    <norm id="\&gpID;nCB20" type="obligation" role="sensor" mission="mPenalties" />
    <norm id="\&gpID;nCB22" type="obligation" role="gateway" mission="mPenalties" />
    <norm id="\&gpID;nCB23" type="obligation" role="corePlfm" mission="mPenalties" />
    <norm id="\&gpID;nCB24" type="obligation" role="application"
    mission="mPenalties" />

    <norm id="\&gpID;nCB30" type="obligation" role="sensor" mission="mBill" />
    <norm id="\&gpID;nCB32" type="obligation" role="gateway" mission="mBill" />
    <norm id="\&gpID;nCB33" type="obligation" role="corePlfm" mission="mBill" />
    <norm id="\&gpID;nCB34" type="obligation" role="application" mission="mBill" />
  </normative-specification>
</organisational-specification>
```