Adaptive Organisation Centered Multi-Agent Systems

Jordi Campos i Miralles

Barcelona, May 2011

Facultat de Matemàtiques - Departament de Matemàtica Aplicada i Anàlisi

Supervisors:

Dr. Maite López-Sánchez (Universitat de Barcelona, UB)
Dr. Marc Esteva (Institut d'Investigació en Intel.ligència Artificial, IIIA, CSIC)

Als meus tresors

"*Il nous appartient de veiller tous ensemble à ce que notre société reste une société dont nous soyons fiers*" from Indignez-vous!, 2010

Stéphane Hessel (1917-),
contributor to the Universal Declaration of Human Rights in 1948.

Agraïments

Un treball com aquest és fruit de molt d'esforç, i aquest esforç no és només d'un individu. Així doncs, es tracta d'una fita personal, i alhora, d'un assoliment col·lectiu. Les analogies amb el contingut social de la tesi són doncs, nombroses.

Començo agraint les hores dedicades als meus dos tutors de tesi, la Dra. Maite López Sánchez i el Dr. Marc Esteva. Ambdós han aportat el seu coneixement en la matèria, però també la seva experiència en la metodologia d'una feina d'aquesta envergadura. Més enllà del terreny professional, recordaré com la Maite va contribuir a la canastreta de la meva filla i en Marc als plantejaments a mig termini.

Per extensió, també he d'agrair els mitjans de les dues institucions que hi ha al seu darrere, la Universitat de Barcelona (UB) i l'Institut d'Investigació en Intel·ligència Artificial (IIIA). D'una banda, la UB ha estat el meu lloc de treball durant els darrers sis anys, i m'ha donat la possibilitat de gaudir fent classes i conèixer a savis com els meus companys d'assignatura, de despatx i de tertúlies de sobretaula. No vull posar els noms, per no fer la llista massa llarga i córrer el risc de deixar-me algú, però una abraçada a tots/es vosaltres. D'altra banda, l'IIIA ha estat el meu contacte més directe amb la recerca professional, on he tingut l'oportunitat de conèixer aquest món en profunditat. Recordaré doncs, molts dels que heu fet interessants i agradables les meves visites a l'institut.

Tot plegat, però, no voldria deixar de mencionar la Universitat Politècnica de Catalunya (UPC), on vaig començar el doctorat. Animat per en Sebas, vaig embarcar-me a la recerca de la mà de la Dra. Daniela Tost i la Dra. Anna Puig. A ambdues els dec un merescut agraïment per aquest inici. Més encara, gràcies Anna per continuar present fins al final d'aquest treball. Tampoc hi ha lloc per esmentar la resta de gent amb qui vam assistir a classe, dinar plegats o simplement xerrar extensament pels passadissos.

Tan mateix, més enllà de l'àmbit de treball, hi ha persones molt estimades que han contribuït extensament al fet que pogués dedicar el temps i els ànims a aquesta tesi. Present dins meu, l'Olga sempre m'ha donat suport i ha tingut cura del nostre tresor. He de confessar que, com a mínim, un tros de la tesi i del meu cor els pertoca a elles. També proper, ha estat el caliu de ma germana Ivanna, la meva àvia Felisa, la família més propera, els amics i fins i tot la sogra. Ara bé, he volgut reservar aquest final a l'inici de tot plegat, als meus pares, la Rosalia i en Ximo. Sense la seva estimació, els seus valors, i la cultura de l'esforç que han mirat de transmetre'm des de petit, no hauria arribat fins aquí.

Moltes gràcies a tots/es!

Contents overview

Detailed contents

List of Figures

Abstract

This thesis focuses on the design and development of Adaptive Organisation Centred Multi-Agent Systems (AOCMAS) in regulation-oriented scenarios. In short, Multi-Agent Systems (MAS) are computational systems where a set of autonomous software components (agents) interact within an environment. Within MAS, the Organisation Centred Multi-Agent Systems (OCMAS) have proven to be successful in promoting a coordination model that structures agent interactions. However, changes in agents' behaviour or in the environment may lead to a poor fulfilment of the system goals, and so, its entire organisation needs to be adapted. We refer as Adaptive OCMAS (AOCMAS) to those systems that are able to dynamically adapt their organisational components to better accomplish their goals. In particular, we advocate for endowing the organisation with adaptation capabilities, instead of expecting agents to be capable of adapting the organisation by themselves. Furthermore, we propose a solution based on a regulative approach instead of subtasks. Hence, our solution is able to deal with problems in which there is no goal that can be decomposed into tasks assigned to agents —e.g. improving traffic flow in a road-network.

We argue that existing OCMAS' components are devoted to enact the coordination model mentioned above. Hence, we review the existing OCMAS works as a set of coordination support mechanisms. Even more, from this point of view we envision a set of mechanisms that assist coordination further than just enacting it. In particular, we regard the adaptation of an organisation as an assistance service that provides an added value to merely enabling the organisation existence. For this purpose, we present a formalisation of such an adaptation service and an abstract architecture to implement it, the so-called 2-LAMA.

We exemplify all these concepts in different problems driven by goals. Nonetheless, we illustrate our whole approach by applying it to a case study in a Peer-to-Peer sharing network (P2P) scenario. In particular, we analyse the adaptation of agents' social structure and a set of norms —both of them are organisational components. Moreover, due to the complexity of such processes, we study two alternatives to perform the latter. That is, we adapt norms by using a heuristic approach and a machine learning technique. Specifically, we use an extension to Case-Based Reasoning machine learning technique that is able to deal with multidimensional continuous search spaces, unknown optimal solutions and awkwardness to identify the impact of a solution on the final result.

A relevant effort of this thesis has been to develop a simulator that enables to compare different coordination models in the P2P case study. It provides several tools to extract and analyse measures from communication network level up to agent level. Furthermore, it has a modular mechanism to easily test different norm adaptation approaches. We use it to empirically evaluate our proposals. The results show that the cost of introducing an additional layer in charge of the system's adaptation is lower than its benefits.

Resum

Aquesta tesi es centra en el disseny i desenvolupament de Sistemes Multi-Agent Adaptatius Centrats en l'Organització (AOCMAS) en escenaris regulats no orientats a tasques. En síntesi, els Sistemes Multi-Agent (MAS) són sistemes de programari, on un conjunt de programes autònoms (agents) interactuen en un entorn. En particular, els MAS Centrats en l'Organització (OCMAS) han demostrat ser efectius alhora de promocionar un model de coordinació que estructuri les interaccions dels agents. Tan mateix, els canvis en el comportament dels agents o l'entorn poden decrementar la seva l'efectivitat, i fer-se necessària l'adaptació de l'organització. Anomenem OCMAS Adaptatius (AOCMAS) als sistemes que són capaços d'adaptar dinàmicament la seva organització per assolir millor els seus objectius. En particular, som partidaris de dotar l'organització amb les capacitats d'adaptació, enlloc d'esperar que els agents siguin capaços d'adaptar-la. És més, proposem una solució basada en la regulació d'interaccions enlloc del repartiment de tasques. Així, aquesta és capaç de fer front a problemes en els quals no hi ha una fita que sigui viable dividir en subtasques executades pels agents —p.ex. millorar el flux de trànsit en una xarxa de carreteres.

Pensem que els OCMAS existents vetllen per fer possible el model de coordinació esmentat. Per tant, entenem que els components d'aquests OCMAS són mecanismes de suport a la coordinació. De fet, amb aquesta perspectiva proposem una seguit de mecanismes que ajudin a millorar aquesta coordinació, més enllà de simplement permetre-la. En particular, considerem l'adaptació d'una organització com un servei d'assistència que proporciona un valor afegit a la organització en si. Amb aquesta finalitat, es presenta una formalització d'aquest tipus de servei d'adaptació i una arquitectura abstracta per a poder-lo proporcionar, que anomenem 2-LAMA.

Al llarg de la tesi, exemplifiquem aquests conceptes en diversos problemes que es guien per objectius que no és poden dividir en subtasques. En particular, el nostre cas d'estudi és una xarxa d'intercanvi de dades (P2P). En aquest escenari, s'analitza l'adaptació de les relacions socials i d'un conjunt de normes —ambdós són components de l'organització. Donada la complexitat d'aquests processos d'adaptació, s'estudien dues alternatives pel que fa a l'adaptació de normes: usant una heurística o bé usant aprenentatge automàtic. En concret, la tècnica d'aprenentatge automàtic és una extensió del Raonament Basat en Casos (CBR), que és capaç de fer front a espais de recerca continus multidimensionals, quan es desconeix la solució òptima i és difícil d'identificar l'impacte d'una solució en el resultat final.

Una part important de recursos s'han dedicat a desenvolupar un simulador que permet comparar els diferents models de coordinació a l'escenari P2P. Aquest simulador ofereix diverses eines per extreure i analitzar mesures tant de l'activitat dels agents, com de la seva xarxa de comunicacions. La seva arquitectura modular facilita la incorporació i comparació de mecanismes d'adaptació. De fet, l'hem usat per avaluar empíricament les nostres propostes i hem obtingut resultats que mostren que el cost d'introduir una capa a càrrec de l'adaptació del sistema és menor que els beneficis que proporciona.

Chapter 1

Introduction

In this chapter, we describe the motivation of this thesis, we present the initial objectives, and we describe the structure of this dissertation. Briefly, this thesis focuses on the design and development of Adaptive Organisation-Centred Multi-Agent Systems (AOCMAS) in regulation-oriented scenarios. The underlying Organisation Centred Multi-Agent Systems (OCMAS) have proven to be successful in promoting a coordination model that structures agent interactions. However, changes in agents' behaviour or in the dynamics of the environment may lead to a poor fulfilment of the system goals, and so its entire organisation needs to be adapted. We refer as Adaptive OCMAS (AOCMAS) to those systems that are able to dynamically adapt their organisational components so to better accomplish their goals. In particular, our main objective is endowing the organisation with adaptation capabilities, instead of expecting agents to be capable of adapting the organisation by themselves. For this purpose, this dissertation reviews previous MAS works, suggests our AOCMAS approach and empirically tests it.

1.1 Motivation

We can describe a Multi-Agent System (MAS) as a set of distributed autonomous entities (agents) that interact within an environment to achieve their common and/or individual goals [Jennings et al., 1998]. On the one hand, such an environment can be defined by its general context characteristics (e.g. spatial divisions), state (e.g. time), and/or existing objects (e.g. resources). On the other hand, such interactions can be collaborative, competitive or both at the same time. Henceforth, we refer to them as *coordination* which we assume is structured by means of an underlying *coordination model* (e.g. interaction protocols are an explicit coordination model). Even more, we argue that MAS infrastructure and methodologies help to promote such coordination model among agents. Thus, we regard these infrastructures and methodologies as a set of mechanisms that provide what we call a *coordination support*.

The overall structure of such interactions (i.e. the coordination model) may be explicitly designed –in OCMAS [Ferber et al., 2004]– or may emerge implicitly as a result of agent activities —in Agent Centred MAS approaches, ACMAS [Serugendo et al., 2006]. In particular, OCMAS approaches use explicit regulative entities called organisations [Horling and Lesser, 2004]. An *organisation* constrains the system evolution and allows agents to construe other participants' behaviour by considering organisational components such as social conventions or enacted roles. Thus, they help designers to predict/regulate the system evolution within certain bounds. This is specially relevant in open MAS [Hewitt, 1986], populated by heterogeneous agents that are developed by third-parties and that may enter or leave the system at any moment. Thus, there are no guarantees about agents behaviour, and so, MAS openness without some regulations may lead to chaotic behaviours. In fact, agent organisations are inspired by human real-world organisations, which have also proved useful in structuring open human societies.

However, changes in the environment or the agent population may decrease the ability of an organisation to fulfil its goals. Thus, adapting such an organisation has been studied for more than a decade now [Avouris and Gasser, 1992, Costa and Demazeau, 1996, Horling et al., 2001, Hübner et al., 2004, Guessoum et al., 2004] and it still constitutes a major research topic [Kota et al., 2009, Sims et al., 2008], since it can help to obtain the expected outcomes under changing circumstances. This adaptation is aligned with the computational organisational theory, which claims that the best organisation designs are domain –and context– dependent [Carley, 1995]. From an ACMAS perspective, organisational changes are expected to emerge from agents' activity. In contrast, OCMAS reason about the organisation to adapt it in order to induce changes in agents' activity. Henceforth, we refer to these latter systems as *Adaptive OCMAS* (AOCMAS). When dealing with such systems, some research questions arise: how the organisation is specified; to what extend it influences agents' behaviour; who defines it; how it can be adapted; who is in charge of adaptation; when adaptation occurs; how to deal with transition periods; and what is the cost of change adoption. The answers to previous questions depends, among others, on which kind of problem is solved and which sort of approach is taken. Hence, next section establishes both issues –the sort of problems and approaches– for the rest of this thesis.

1.2 Problem statement

In this thesis, we focus on problems driven by goals which are not decomposed into subtasks that agents commit. Alternatively, these goals are usually expressed as expected system performance measures instead of system tasks. Accordingly, existing approaches are not based on assigning subtasks to agents, but they use a normative system to regulate agents' activity. In this section, we first define such problems and identify some of the approaches that can deal with

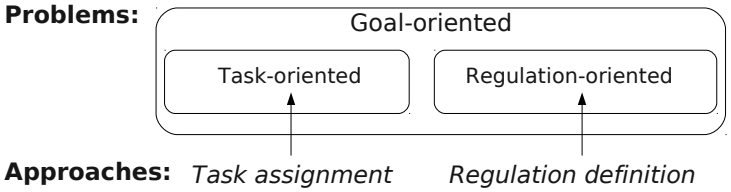


Figure 1.1: Goal-oriented problems.

them. Then, we present three scenarios that pose this sort of problem. Such scenarios are used to illustrate the problem described in this section but also to exemplify several concepts along all this document.

1.2.1 Regulation-oriented problems

Within OCMAS, we distinguish between different approaches when dealing with problems driven by goals —which we name *goal-oriented problems*. Figure 1.1 depicts these approaches and the corresponding sorts of problems. On the one hand, there is an approach we call *task assignment approach*, which identifies tasks which accomplish system goals, decomposes them into subtasks, and assigns them to agents. As a consequence, it assumes that agents committing to tasks need to somehow incorporate task execution into their individual goals. Hence, this approach can only deal with goal-oriented problems where system goals can be translated into tasks that agents assimilate. We denominate these problems *task-oriented problems*. Currently, most of previous work on adaptive OCMAS [Hübner et al., 2004, Lesser et al., 2004, Guessoum et al., 2004, Zhang et al., 2009], suggests to re-assign new tasks to agents when circumstances change. That is to say, these previous works are task assignment approaches applied to task-oriented problems.

On the other hand, there is an approach we call *regulation definition approach*, which consists on setting some regulations that bound agents' activity while preserving much of their autonomy —instead of assigning subtasks to them. However, establishing the mapping between regulations and goal accomplishment is a complex issue, and it is still harder to know how to adapt these regulations when organisational goals are not being achieved. Notwithstanding this complexity, as this approach assumes individual and system goals are loosely coupled –or even independent–, it can deal with problems where system goals cannot be decomposed into tasks that agents assume. We term these problems *regulation-oriented problems*.

In particular, in this thesis we focus on scenarios that pose this last sort of problems. Furthermore, we specifically focus on problems that have the environmental characteristics described next —the most of them are related to well-known environment attributes [Russell et al., 1995, Wooldridge, 2009]. We aim to deal with environments that are *inaccessible*, so it is not possible to observe all the factors that influence system's evolution. This partial information

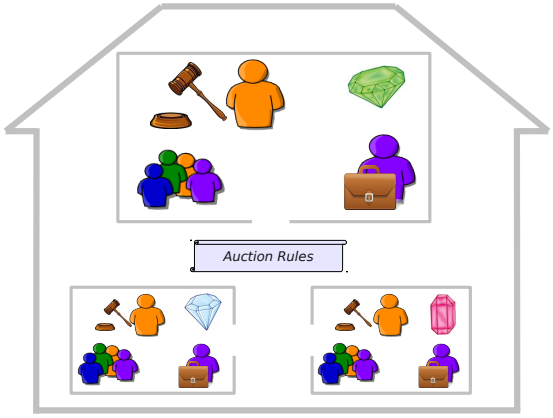


Figure 1.2: Auction House scenario.

make them appear to be *non-deterministic*, so a given action executed in a state does not always result into the same next state. Furthermore, the environments we consider are *dynamic*, since, for an agent, the environment may change even if this agent is not performing any action. This is due to environmental processes and other agents acting simultaneously. Additionally, we focus on environments that are *continuous*. That is, they may not have neither a discrete number of states nor a discrete number of actions. Finally, when adapting their regulative structures, we assume these environments are *run-time adapted* because their activity continues even while these structures are being updated.

1.2.2 Illustrative scenarios

In order to illustrate the problem stated above, we present three scenarios that pose this sort of regulation-oriented problems: an auction house, a traffic scene and a Peer-to-Peer sharing network. These same scenarios are also used along the rest of the document to exemplify different concepts. Even more, the last one is the case study of this thesis.

1.2.2.1 An auction house

The *auction house scenario* depicted in figure 1.2 consists in different activities where different goods (e.g. the jewels in the figure) provided by sellers (the men with a briefcase) are auctioned by auctioneers (the men with a hammer) according to some auction rules (a rolled paper). In such a scenario, the system goal of increasing the amount of transactions cannot be decomposed into task assimilated by sellers and bidders (the group of men) due to their private competitive goals. Alternatively, a set of regulations –auction rules– act as an coordination mechanism that help to accomplish global goals. Besides, we assume this scenario has an inaccessible non-deterministic environment. Because,

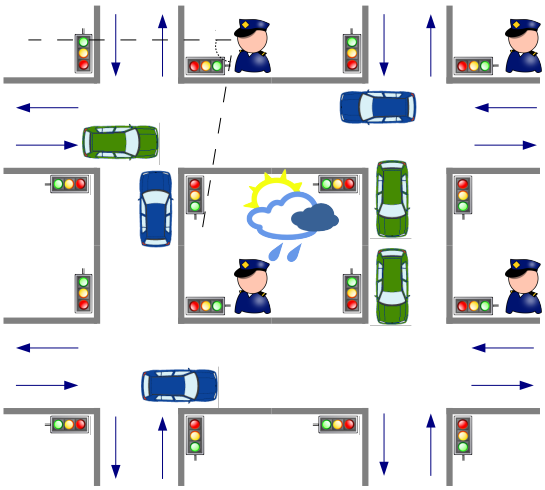


Figure 1.3: Traffic scenario.

due to mentioned privacy issues, it is not possible to know bidders' strategy. So, it is not possible to determine which is going to be the result of a particular auction. Moreover, from the point of view of a particular agent –e.g. a bidder– the environment is dynamic since it may change even if the agent is not performing any action —e.g. a product might be sold while the bidder is deliberating about its next bid. Above all, when there are changes in the participants or the goods, adapting the regulations may be convenient to enhance system performance. For instance, if the incoming products are perishable, the auction rules may switch from English to Dutch auction protocol to shorten the auction time.

1.2.2.2 A traffic scenario

The *traffic scenario* consists on a grid of streets populated by circulating cars as depicted in figure 1.3. In this scenario, the individual agent goals (i.e. reaching a car's destination as fast as possible) may not be aligned with global goals (having a fluid traffic flow). Hence, these goals cannot be directly mapped into car' basic driving tasks since we assume drivers (cars in the figure) do not reveal their private destinations nor accept driving plans. Instead, there is a set of regulations (e.g. the traffic lights) as coordination mechanisms that help to accomplish global goals. These regulations can be updated by police agents (policemen) depending on their observations. For instance, they can observe the traffic flow in their road-network region (exemplified as a pair of vision dashed lines) or environmental factors (e.g. the weather). Regarding the scenario's environment, we assume it is inaccessible and non-deterministic because cars destinations are unknown due to privacy issues and can change at any moment. Hence, it is not possible to determine the resulting traffic flow after a specific update of traffic lights' intervals. Even more, it is dynamic because while a

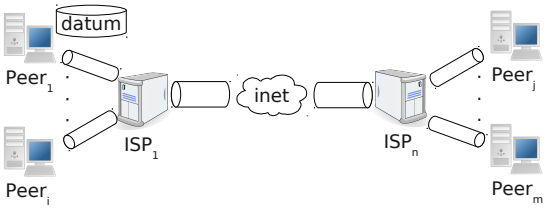


Figure 1.4: P2P scenario.

policeman is deliberating about giving priority to a certain driving direction –e.g. by updating some traffic lights' intervals–, the traffic flow may change —due to flow dynamics, cars' destination updates or weather factors. Besides, we consider this environment is continuous since car positions and speed are not discrete. Finally, when regulation structures are updated, we assume this environment is run-time adapted. For instance, when a policeman tries to improve traffic flow by updating traffic lights' intervals, he requires flow observations that may vary while he is deliberating.

1.2.2.3 A P2P Sharing network

The *Peer-to-Peer sharing network scenario* (P2P henceforth) is our case study. Hence, in addition to illustrate different concepts in this scenario, we also apply our whole approach, implement it and perform an empirical evaluation in next chapters. For now, we simply provide a brief description like we have done with previous scenarios.

This scenario is composed by a set of computers (called peers) that contact among them to share some data as depicted in figure 1.4. They communicate by accessing the Internet through their Internet Service Providers (ISP). Such peers can be regarded as agents developed by third parties with the individual goal of obtaining the data as soon as possible. Also, the system goal is to spread the data among all agents as soon as possible. However, if all agents try to get the data at the same time, the network may become saturated. Then, even if some peers achieve their individual goal soon, the overall process will be longer, so the system goal will be achieved later. Hence, the individual and system goals may not be aligned depending on network topology and status. Ideally, a peer having the datum (so-called seed) should send the data only to a subset of the peers lacking it (so-called leeches) in order to avoid net saturation. This action could be regarded as a subtask, so a task-decomposition approach could be taken. However, due to the open nature of peer-to-peer sharing networks, it is not possible to impose tasks on participants and expect that they commit such tasks. Hence, P2P sharing networks are usually based on some regulations (such as protocols and norms) to bring the system towards its global goal while keeping most of agent's autonomy. In other words, they use a regulation definition approach.

In addition, when there are changes in the network status or the agent population, such regulations can be adapted to keep system performance. In such a case, we assume P2P's environment is run-time adapted since peer computers do not stop exchanging messages while these regulations are changed. Moreover, we argue this environment is inaccessible, non-deterministic and dynamic. It is inaccessible because the details about Internet traffic are unavailable. Consequently, message latencies cannot be predicted, so it is non-deterministic. Even more, these latencies may change while computers choose their fastest partner (the one they choose to request the datum), so the environment is dynamic. Also, this environment has continuous state and action spaces since we assume that neither the network saturation measures nor computer's network consumptions are discrete. Chapter 6 provides further details about these network measures, the underlying network abstraction and the scenario in general.

1.3 Objectives and Contributions

The initial aim of this thesis was to review previous work from our coordination support point of view, in order to detect some potential research area and suggest a MAS enhancement from this perspective. The result is an abstract MAS architecture proposal, that is able to assist agents to improve system performance.

For this purpose, we introduced the following objectives:

1. to define the concept of *coordination support* related to sustain a coordination model among MAS participants.

2. to review the *previous work* from this perspective in order to identify some areas that could be enhanced.

3. to propose an *approach* to provide assistance services to agents.

4. to consider different *alternatives* to provide such assistance in the form of organisational adaptation.

5. to *evaluate* our approach in a P2P sharing network scenario.

In particular, our first objective (1) was to define the concept of *Coordination Support* related to sustain a coordination model among MAS participants. This vision was inspired in our understanding of MAS infrastructure evolution. Initially, a MAS was designed *ad hoc* without any special methodology, developing its own infrastructure from scratch [Jennings et al., 1998]. However, as MAS area evolved, certain tasks were abstracted and gradually provided by MAS infrastructure as domain independent services. These services alleviate agent development, since participants can use them instead of doing the corresponding tasks. Furthermore, these services help to structure agent interactions. Thus, we call them Coordination Support mechanisms. They include from simple individual services that depend on a single agent, to complex collective services

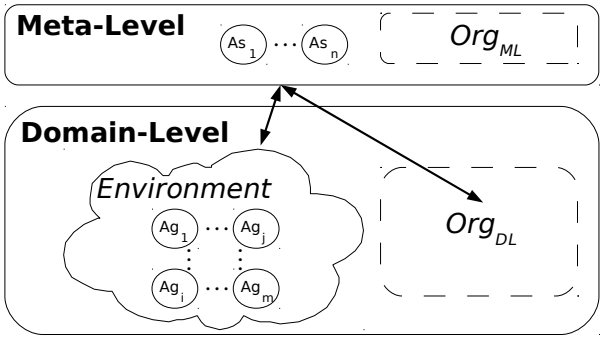


Figure 1.5: 2-LAMA architecture.

that require information related to a set of agents. Accordingly, we group these services in different subsequent layers.

Next, as the second objective (2), we planned to review the previous work from this coordination support perspective. During this process we identified services that range from connectivity issues concerning data exchange to organisational aspects related to social structure or rule enforcement. Moreover, we even found some services that fit into our proposed top coordination support layer: the *Assistance Layer*. We argue these services assist coordination rather than merely enabling it. In other words, instead of "making coordination happen", these services "help to coordinate seamlessly". This functionality is similar to ambient intelligence [Vallee et al., 2005], where the infrastructure supports human activities, but in this case it is focused on assisting MAS participants. This layer may even have pro-active capabilities that let the MAS infrastructure take the initiative and act intelligently —e.g. providing advices to agents that let them take more profit of current coordination mechanisms. Within these capabilities, we see the *organisational adaptation* as a coordination support service present in AOCMAS. This service is particularly useful when environmental or agent population changes avoid the organisation achieving its goals.

The third objective (3) was to propose an approach to provide such assistance services to a MAS. Specifically, our objective was to propose a MAS abstract architecture that could deal with the defined regulation-oriented problems. We started with an approach called *Situated Autonomic Electronic Institution* (SAEI). It was the formalisation of the existing Autonomic Electronic Institution approach [Bou et al., 2006] and its extension to deal with existing MAS —the original approach was restricted to Electronic Institutions [Esteva, 2003]. Its main drawback was that it was a centralised approach and it was strictly focused on providing an organisational adaptation service. Alternatively, we later proposed the *Two Level Assisted MAS Architecture* (2-LAMA), which is distributed and was conceived considering different assistance services. Our main intuition to conceive this architecture is depicted in figure 1.5. Following the meta-level abstraction [Corkill and Lesser, 1983], we advocate for adding a

meta-level that provides different assistance services to a domain-specific MAS, which we call the *domain-level*. Specifically, we focus on a particular assistance service, the organisational adaptation. That is, the meta-level agents (so-called *assistants*, As_i in the figure) are organised (Org_{ML}) to adapt the organisation (Org_{DL}) of the agents participating in the regular domain activity (Ag_i). These assistants are meant to reason at a higher level of abstraction than domain agents. Following a division of labour paradigm, instead of increasing the complexity of domain agents, assistants are the ones in charge of taking into account organisational goals. Moreover, assistants act as trusted third parties, since they are not involved in domain activities and they are provided by the infrastructure.

The fourth objective (4) was to consider different alternatives when designing the meta-level approach to update domain-level organisation. Hence, we started by dealing with the adaptation of two different organisational components: the relationship among agents and a set of norms (i.e regulations expressed in deontic logic). Such adaptation requires a knowledge about the relationship between these components and system performance. This adaptation knowledge can be provided by system designer at design time (e.g. by coding a heuristic) or gained at run time by using machine learning. The former requires previous expert knowledge and provides a fixed estimation. Whereas the latter estimates this relationship automatically and is able to evolve it along time. Accordingly, we designed a heuristic to adapt each organisational component (agents relationships and norms), but also we took a machine learning approach when dealing with the adaptation of the latter component (norms). Specifically, we proposed a tailored Case-Based Reasoning (CBR) [Riesbeck and Schank, 1989] that is able to deal with multidimensional continuous search spaces, unknown optimal solutions and the awkwardness to identify the impact of a solution on the final result.

Finally, the last objective (5) was to evaluate our approach in the peer-to-peer sharing network scenario (P2P). Nowadays, it is a relevant application thanks to the Internet expansion. In addition to this fact, it is a scenario that poses a regulation-oriented problem like other usual MAS scenarios —such the auction house and traffic scenarios presented previously. In order to perform an empirical evaluation on this scenario, we developed a simulator that gives us control over all details even at communication network level. This is important, since the communication aspects are relevant in our case study —like network latencies. For instance, the simulator lets us access to detailed network statistical information that is not available in the Internet[1] or it lets us determine which is the status of network links at the beginning of a test. We use this simulator to perform several executions while using different adaptive alternatives. Moreover, as base-line, we also compare our proposals to a non-adaptive approach widely used in P2P, the BitTorrent protocol [BitTorrentInc., 2001]. Even more, we implemented some open MAS extensions in the simulator and performed some

[1] Notice that this statistical information is available to the experiment designer, so it can be used to analyse system behaviour. However, MAS participants cannot access it, like in the real scenario.

exploratory experiments. These additional experiments provided positive results of our approach when dealing with entering/leaving agents and norm-violator agents.

1.4 Structure

The rest of this thesis is divided in the following chapters:

Chapter 2 State of the art: it surveys *previous work* on topics relevant to this thesis from our Coordination Support point of view. This chapter is divided in several sections that define the *Coordination Support* term structured in layers, and reviews existing works according to these layers.

Chapter 3 SAEI: it goes further reviewing the *Situated Autonomic Electronic Institutions* (SAEI), which is the antecedent of our OCMAS abstract architecture proposal. It performs an organisational adaptation of an existing MAS by means of an overlapped Electronic Institution.

Chapter 4 Assistance Layer: it focuses on the upper Coordination Support layer, the *Assistance layer*. It defines the group of services devoted to assist agents (Agent Assistance) and the adaptation service related to organisational structures (Organisational Assistance).

Chapter 5 Organisational Adaptation: it goes further detailing the *Organisational Adaptation* service presented previously and proposes how to provide it. First, this chapter formalises the organisational model, and the features related to its adaptation. Next, it presents our enhanced approach, the *Two Level Assisted MAS Architecture* (2-LAMA).

Chapter 6 Case Study: it applies 2-LAMA to the Peer-to-Peer sharing network *case study* (P2P). It firstly defines all details of the P2P scenario and afterwards it specifies a MAS within this domain according to 2-LAMA model.

Chapter 7 Adaptation mechanisms: it proposes how to adapt two different organisational components in the P2P scenario. On the one hand, it describes a heuristic *to adapt the relationships among agents*. And, on the other hand, it presents two alternatives *to adapt norms*: a heuristic approach and a machine learning approach. The latter is an extension to Case-Based Reasoning (CBR) that suits the characteristics of the P2P case study.

Chapter 8 P2P MAS Simulator: it describes the *simulator* developed to test different alternatives in the P2P sharing network scenario. The chapter details its internal architecture, the general simulation cycle and the analysis facilities it provides.

Chapter 9 Experiments: it presents the *experiments* performed to empirically evaluate our approach. It describes the different coordination models –i.e. approaches– contrasted, the design of the experiments to evaluate them and the results analysed.

Chapter 10 Conclusions: it discusses the contributions of this research and how it can be extended in the future.

Chapter 2

State of the art

Along this thesis, we argue that MAS methodologies and infrastructures help to structure agent interaction. In fact, we claim they help to design and execute a coordination model among agents. Hence, we regard them as a set of mechanisms that provide what we call a *Coordination Support*. In particular, in this chapter we define this concept structured in layers and we review the *related work* under this perspective in order to settle some basic concepts that are used in the rest of the thesis.

Notice that when describing these works, we explain their approaches using our coordination support nomenclature. However, we also provide the terms they use to refer to the same concepts enclosed in brackets. Even more, most of the figures we include, are taken from those works. Hence, in addition to present our coordination support vision, we aim to facilitate understanding these approaches.

2.1 Coordination Support

We regard a MAS as a set of agents interacting within an environment to achieve some goals [Kephart and Chess, 2003]. This interaction can be direct or indirect. For instance, a direct interaction may be a message among two agents, whereas an indirect interaction may be that a resource is not available to an agent when another one is using it. In both cases, we say these agents are *coordinated* because their interaction reflects that they affect or are affected by actions of other participants. In fact, we interpret previous definitions of the coordination concept in a wide sense: coordination is the integration of different activities to obtain a certain dynamic [Omicini et al., 2004], avoiding harmful interactions and promoting beneficial ones [Jennings et al., 1998], from a system-wide perspective. This global perspective lets us talk about coordination in cooperative and competitive scenarios with individual and collective goals.

Above all, such interactions are structured according to a *coordination model* defined at design time. In fact, current tools and methodologies generally use

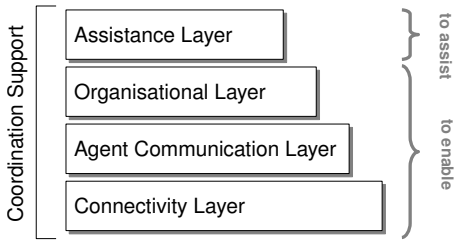


Figure 2.1: Coordination Support layers.

an infrastructure to provide services that aid agents to enact this coordination model. Hence, we use the term *coordination support* to denote those services that are useful for agent coordination.

Moreover, we conceive these services structured in layers as depicted in figure 2.1. We conceived this structure to encompass existing MAS works, taking into account previous stratified divisions such as the one from the Foundation for Intelligent Physical Agents (FIPA Abstract Architecture [FIPA, 2002]), the one appearing in Tuple Centres Spread over Networks (TuCSoN [Omicini et al., 2004]) or the one from RETSINA [Sycara et al., 2003]. As in previously cited approaches, in our coordination support perspective each layer provides functionalities required by its subsequent one. This abstraction simplifies agents' engineering and reduces the overall engineering complexity by isolating each functionality. In particular, we distinguish three basic layers devoted to enable agents coordination and a last layer devoted to enhance this coordination by assisting participants. The bottom layer (Connectivity Layer) provides low level communication functionalities —e.g. defining a physical connection. On top of it, a distinct layer (Agent Communication Layer) supports knowledge exchange among agents by means of a communication language —e.g. defining a message structure. And above, we see a distinct layer (Organisational Layer) that provides another relevant coordination component, the organisational aspects —e.g. defining roles.

Furthermore, we advocate for considering an additional layer (Assistance Layer) on top of previous ones, that provides assistance to fulfil the coordination model enabled by the other layers. That is to say, this layer aids agents to use more effectively and efficiently previous coordination mechanisms. We see this layer as a step forward in MAS development, since it could facilitate the engineering and enrolment of heterogeneous agents —specially relevant in open MAS[1], where agents are designed by different parties.

Next sections illustrate each enabling layer by reviewing existing works under our coordination support perspective. In addition, due to the novelty of the Assistance Layer concept, it is addressed in an additional chapter 4, which also includes some references to related work.

[1] An open MAS is an open system [Hewitt, 1986] where there is no control over the agents design process, so their behaviour is totally unknown. Accordingly, they can join and leave in any moment or even try to transgress MAS social conventions.

2.2 Connectivity Layer

The first requirement to enable agent coordination is to allow the exchange of information among them. This is covered by what we call the *Connectivity Layer*, which usually provides:

- a *physical connection* specification
- a *protocol* to use this connection
- a reliable *transport service*

Even more, it may also provide:

- a *data encoding* specification
- an *addressing format*
- an *exception handling mechanism*

Most of these functionalities can be covered by using the existing network standards such as the Open Systems Interconnection Basic Reference Model (OSI Model) [ISO 7498-1, 1994]. This OSI Model provides an abstraction of the interconnectivity among different systems, and there are several standard specifications covering specialised parts of it. For instance, FIPA [FIPA, 2002] suggests the following alternative communication standard protocols in their Agent Message Transport Specification [FIPA, 2001d]: Hyper-Text Transfer Protocol (HTTP), Internet Inter-Orb Protocol (IIOP) or Wireless Application Protocol (WAP).

As an illustration, the Java Agent DEvelopment Framework (JADE [Bellifemine et al., 2007]) is one of the available implementations of FIPA Specifications, so it uses previously cited standards. Recently, another FIPA compliant implementation, Smart Python multi-Agent Development Environment (SPADE [Gregori et al., 2006]), uses the Internet instant messaging protocol Extensible Messaging and Presence Protocol (XMPP), as a transport protocol. This protocol offers new features like Presence Notification and Multi-User Conference, and has been proposed to be added to the FIPA Specifications [Cámara et al., 2006].

2.3 Agent Communication Layer

Further than exchanging data –as supported by previous layer– agents actually exchange knowledge. Hence, we call *Agent Communication Layer* to the group of services that provide agents with means to exchange knowledge. Such services normally include:

- a *message structure* specification
- a shared *ontology*

- a content *language*
- a *directory service* to locate destination agents

The first three requirements are covered by Agent Communication Languages (ACL), which are generally based on the *speech act theory* [Searle, 1969]. This theory argues that communication can be construed as actions that senders make just by uttering messages. In other words, in some cases an agent illocution is an action by itself —it is called *a performative utterance*. For instance, in the auction scenario (see §1.1), when an auctioneer says "Sold!", the good changes its status from being an offer to be a sold product. Hence, further than exchanging knowledge, agents are somehow acting when exchanging messages among them. For instance, the FIPA Communicative Act Library [FIPA, 2001e] defines twenty-two communicative acts like informing, requesting, agreeing or cancelling.

Examples of mentioned ACL are FIPA-ACL [FIPA, 1997, FIPA, 2001a], the Knowledge Query and Manipulation Language (KQML [Finin et al., 1997]) and the Agent Communication Language for Multimedia Communication (ACLMC [Gou et al., 2007]). All of them basically define a set of keywords –so-called *performatives*–, their parameters —e.g. sender, receiver or language– and the actual content. They frequently use a LISP-like syntax to join all these elements. Besides, they offer alternatives to express the associated content, such as the Semantic Language (FIPA-SL [FIPA, 2001i]) or the Knowledge Interchange Format (KIF[2] [KIF, 1995]).

These languages require the usage of a shared vocabulary which is defined by the *ontology component*. For instance, FIPA specifies an Ontology Service [FIPA, 2001h] to perform such functionality, and the World Wide Web Consortium (W3C) has its Web Ontology Language (OWL 2) [Grau et al., 2008] recently revised.

The Agent Communication Layer also provides an *agent directory service* to locate participant agents. This directory stores a relation among agent unique identifiers and their current transport addresses —like the the FIPA Directory Facilitator [FIPA, 2001c]. Thus, agent locations are transparent to agents that send messages. As an illustration of all stated services, JADE implements all cited FIPA specifications. Alternatively, Java Agent Template Lite (JATLite) [Jeon et al., 2002] implements KQML, and S-Moise+ [Hübner et al., 2005] implements both of them.

2.4 Organisational Layer

Like humans, agents achieve a higher level of coordination when working in groups [Frank Dignum, 2000]. Even more, from a system point of view, a degree of organisation among agents help to achieve certain collective properties despite

[2]Recently, KIF evolved into an ISO standard called Common Logic [Common Logic Working Group, 2007].

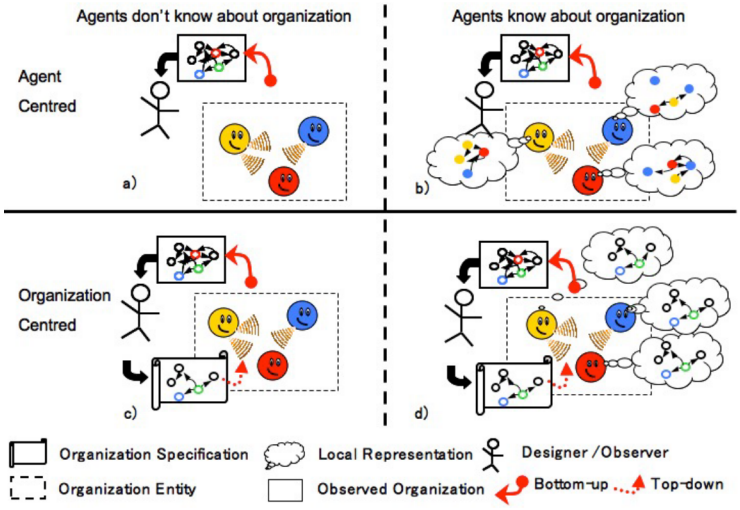


Figure 2.2: ACMAS versus OCMAS from [Boissier et al., 2006].

individual agents' varying behaviour. Thus, in addition to previous services devoted to exchange knowledge among agents, several approaches also count on organisational aspects. These aspects allow to structure the interaction among agents at higher level of coordination than the one provided by previous layers.

Before enumerating such organisational aspects, we distinguish between agent-centred and organisation-centred approaches. Although both may include organisational aspects, our objectives focus on the last ones. Hence, after enumerating related organisational components and services, we review current approaches laying a particular stress on the organisation-centred ones.

2.4.1 Agent-Centred versus Organisation-Centred MAS

The way in which organisational aspects are incorporated into MAS varies among approaches as shown in figure 2.2. As originally stated in [Lemaître and Excelente, 1998], there are two main groups of approaches: Agent Centred MAS approaches (ACMAS) [Serugendo et al., 2006] and Organisation Centred MAS approaches (OCMAS) [Ferber et al., 2004]. In the former (ACMAS), organisations are implicit and may emerge as a result of agent activities. This may happen either if agents don't know about such implicit organisation (figure 2.2.a) or they have their own model about the organisation (figure 2.2.b). In the last case, agents may discover organisational characteristics along the interaction with others or simply they may follow organisational bounds due to its internal codification like in the first case.

In contrast, in OCMAS, organisations are explicitly designed entities that can also be unknown to agents (figure 2.2.c) or be known to them (figure 2.2.d). Besides, as these organisations are generally proposed by the MAS designer,

they help this designer to predict/regulate the system evolution within certain bounds. Moreover, as they are entities on their own, they can persist even when agents enter or leave. Even more, as they are explicit, they may reduce the complexity of programming participant agents since these agents can directly access the organisational specifications and act within their bounds [Dignum and Dignum, 2005]. Furthermore, this same explicit nature facilitates *computational reflection* [Smith, 1982], meaning that a MAS can observe and modify its own structure. For instance, in next layer this feature is used to provide different services based on observing/adapting system organisation.

Above all these alternatives, in this thesis we focus on OCMAS. approaches, in which the organisation is explicit and already exists at the beginning of MAS activity, like in the scenarios described in §1.2.2. Hence, having an explicit organisation, we can refer to its components and we do not need to make assumptions about its participants characteristics. In fact, we focus on those OCMAS that can be adapted, which we call Adaptive OCMAS (AOCMAS).

2.4.2 Components and Services

Before reviewing some current organisational approaches in next section, we summarise the sort of components they usually present and the related MAS infrastructure services. Generally, an organisation may include the following components:

- a *social structure* defined by a set of roles that participants can play and the relation among them
- a set of *social conventions* associated to defined roles
- an *enforcement policy* about these conventions
- some *organisational goals* taken into account when defining previous components

Specifically, a *role* is an abstract description of agent attributes and behaviours. It defines the properties and actions of the agents that play such a role. In fact, the possible actions are delimited by the social conventions, which use roles to refer to involved agents. Moreover, role specification can also include its relations with other roles. These relations may restrict which agents can play a certain role, which other roles can be played simultaneously (compatibility) or what their inheritance relation is. Additionally, composition and authority relations among roles let define social structures such as groups or hierarchies [Horling and Lesser, 2004].

Besides, *social conventions* define what agents should conform and expect others to conform [Lewis, 1969]. This may reduce agent complexity since they can focus on a smaller set of possible actions. These conventions are generally expressed as protocols and norms concerning roles. The former (protocols) define the valid sequences of actions on agent interaction. Whereas the latter (norms), bound agent actions using deontic logic (i.e. defining permissions, prohibitions

and obligations). Such norms are usually called regulative norms compared to so-called constitutive norms [Boella and van der Torre, 2004, Searle, 1995]. Precisely, a constitutive norm defines what count-as a role/action/context in the regulative system, so a regulative norm can use these generic concepts in its definition. For instance, in a traffic scenario, a constitutive norm can define a car as a "vehicle", and another constitutive norm can define crossing the vertical projection of a traffic light in red colour as "to go through a red light". Then, there can be a regulative norm that uses both concepts, like a norm that defines the obligation of a "vehicle" that "goes through a red light" to pay a fine.

Also, in scenarios where agents can violate the above-mentioned social conventions (e.g. in open MAS), the organisation may include an *enforcement policy* that determines how to detect these violations and how to act in such cases. For instance, the detection policies may include no detection, full detection (all agent actions are checked, e.g. using agent wrappers) or partial detection (only some actions are checked, e.g. using police agents). And the consequence policies may include punishment (e.g. fining violators), incentive (e.g. rewarding agents) or imposition (e.g. filtering out improper actions). Nevertheless, the enforcement can also be delegated to participant agents, by using reputation techniques as explained in §2.5.1. A further discussion about norms and enforcement mechanisms can be found in the following norm-related categorisation [Savarimuthu and Cranefield, 2009].

Furthermore, an organisation may explicitly include its goals. These *organisational goals* describe the proposal that guided the organisation design and may differ from participant individual goals —when individual goals are not aligned with the social welfare. Depending on the MAS domain (see §1.2), these organisational goals may be expressed as a global task (in task-oriented problems), or as a set of desired system outcomes (in regulation-oriented problems). In the former case, task assignment approaches divide this global task into subtasks performed by participant agents. In contrast, in the latter case, regulation definition approaches check system observable properties against their desired values. In both cases, these explicit goals can be used to evaluate system performance –depending on the degree of fulfilment of such goals– to determine the extent to which the system is fulfilling its design objectives. Even more, this evaluation can be used to guide the adaptation of the organisation as explained in §2.5.1.

These enumerated services are formalised in §5.3, as part of our contribution to formalise its adaptation. In addition to these components, we can outline the services related to them. In brief, the most of OCMAS approaches provide services to support the specification of organisations, their instantation and the management at run-time. Such services may include:

- storing a list of current engaged agents and their roles
- keeping the state of organisational elements (e.g. protocol states)
- providing an enforcement mechanism (e.g. filtering out improper actions)

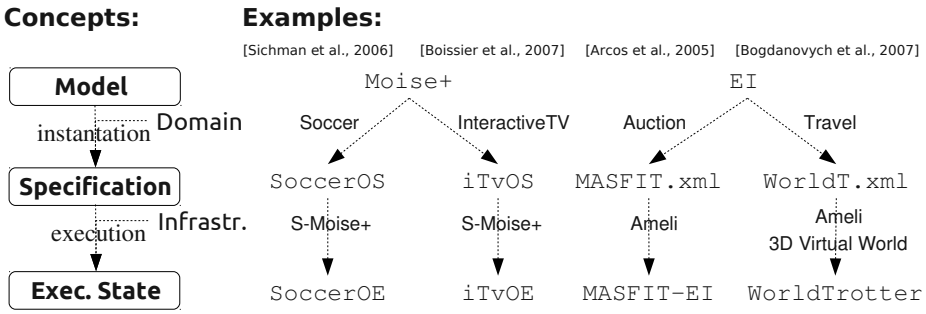


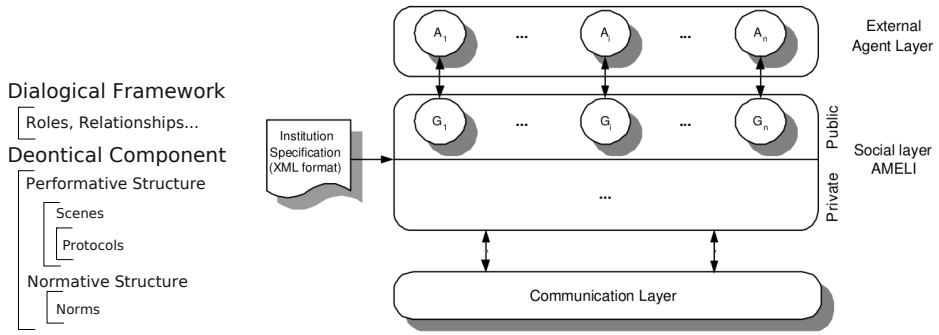
Figure 2.3: Organisation Model, Specification and Execution State.

- supplying a directory to locate services offered by agents (depending on their role/skills)

2.4.3 Current approaches

Currently, OCMAS approaches include an organisational specification and its supporting infrastructure. As described in [Sichman et al., 2006], the former is an specification of the exposed organisation components (Organisation Specification) defined in a modelling language (Organisation Modelling Language), whereas the latter is the infrastructure (Organisation Infrastructure) that interprets such specification and supports its execution (Organisation Entity). Hence, as illustrated in the left part of figure 2.3, we assume that a given organisation *model* can be used to instantiate a particular organisation *specification* for a given domain. Such specification can later be interpreted by a MAS infrastructure and have its corresponding organisation *execution state*. The right part of figure 2.3 illustrates such concepts citing some previous works [Sichman et al., 2006, Boissier and Gâteau, 2007, Arcos et al., 2005, Bogdanovych et al., 2007].

For example, an *Electronic Institution* (EI) [Esteva, 2003] is an organisational specification that is supported by the Ameli infrastructure as depicted in figure 2.4. Its definition includes roles and social conventions. These roles may have compatibility and inheritance relation specifications. Whereas, these social conventions are valid action sequences (protocols in Scenes of its Performative Structure) and their consequences (norms in its Normative Structure). EI's enforcement policy includes full detection, but consequences differ between protocols and rules. Whereas protocol violations are filtered out (i.e. full detection and imposition), norm violations create deontic consequences (e.g. an obligation to pay a fine, i.e. punishment policy). Accordingly, the EI's infrastructure that interprets such a specification (Ameli, see figure 2.4b) has a mechanism to detect and handle violations. In particular, every agent (A_i in the figure) has a wrapper (its Governor G_i) that detects and filters out any prohibited action. Finally, a basic EI does not have



(a) EI's model (b) EI's infrastructure - Ameli from [Esteva et al., 2004]

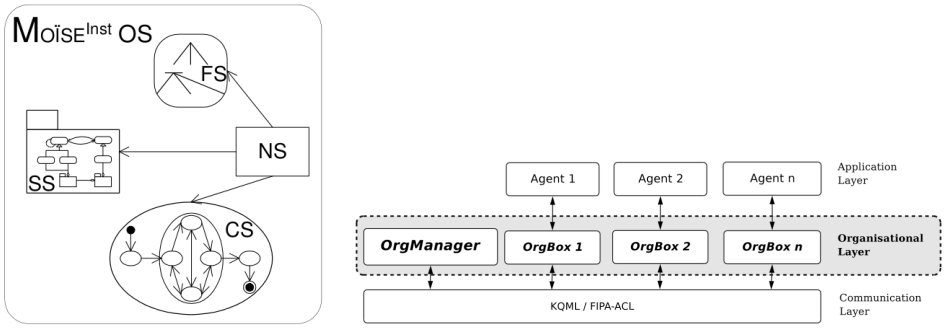
Figure 2.4: Electronic Institution's organisation model and infrastructure.

explicit organisational goals. However, its extension called Autonomic Electronic Institution (AEI) [Bou, 2009] incorporates explicit goals. Specifically, such goals are defined as desired system outcomes, since an EI is a regulation definition approach. See §3.2 for additional details about EI, and §2.5.2 for further details about AEI. Above all, Electronic Institutions are usually conceived as a pure OCMAS approach. However, they could be designed and specified by the agents themselves upon their agreement [Gaertner et al., 2009], which is a hybrid approach that introduces ACMAS principles. In fact, there are also other explicit organisations created and destroyed by agents at runtime [Foster et al., 2004, Cardoso and Oliveira, 2004]. A further discussion about these organisational paradigms can be found in [Horling and Lesser, 2004].

Following a similar scheme, *Moise* [Hübner et al., 2002] has an organisation specified in Moise+ model, which is supported by the S-Moise+ infrastructure as depicted in figure 2.5. It organisational specification (OS) has a role definition very similar to EI's, but it also supports composition and authority relations (in its Structural Specification, SS). Likewise, the social conventions are also expressed as interaction protocols (in its Contextual Specification, CS) and rules (in its Normative Specification, NS)[3]. Also, its enforcement policy is totally strict. In fact, S-Moise+ infrastructure [Hübner et al., 2005] has an agent wrapper too (so-called OrgBox). However, the infrastructure differs in the normative management . While in EI the normative component is integrated into the institution, Moise uses a second organisation (Supervision Organisation) to supervise the norm fulfilment in the original organisation (Domain Organisation). Also, Moise differs from EI in the goal specification, because Moise is a task assignment approach. Hence, its goals are expressed as system tasks that are divided into subtasks (Functional Specification, FS).

In both previous approaches, the roles are the basis to model an organisa-

[3]Specifically, norms are included in a Moise+ extension called Moise^{Inst} which is supported by an S-Moise+ extension called Synai [Gâteau et al., 2005].



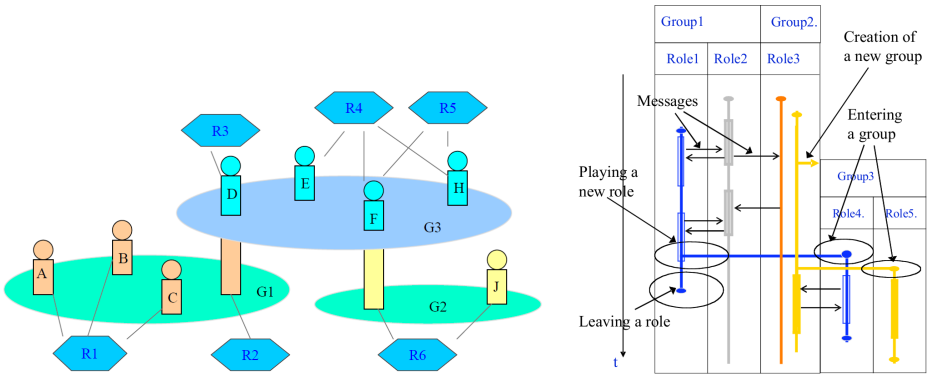
(a) Moise's model (b) Moise's infrastructure

Figure 2.5: Moise's organisation model and infrastructure from [Gâteau et al., 2005] and [Hübner et al., 2005].

tion. In this sense, the *Agent-Group-Role* (AGR) [Ferber et al., 2004] –formerly Aaladin [Ferber and Gutknecht, 1998]– is an organisation model that almost uses only this concept to specify an organisation. figure 2.6a depicts how an organisation is modelled in AGR by specifying different groups (ellipses) in which agents (rectangles) play certain roles (diamonds). Also, it somehow has some basic interaction protocols specifications expressed in a derivative of UML diagrams (so-called Organisational Sequence Diagram, see figure 2.6b). However, it lacks from other organisational components, such as norms or goals. Consequently, the organisational services provided by its organisation infrastructure (the Madkit), consists basically in maintaining a list of group participants and their associated roles.

Besides, there are approaches that offer tools to just model an organisation or only to create a support infrastructure. For instance, the *Prometheus methodology* [Padgham and Winikoff, 2005] is useful to model an organisation without providing an infrastructure to support it. Alternatively, the model is used to create participant agents that behave according to the organisational specification. This specification includes some organisational goals (defined during what they call the initial System Specification phase) and some social conventions defined by protocols (in the Architectural Design phase). Then, this model is used to create agents that follow such conventions (in the Detailed Design phase). Above all, the assumption of controlling agent development, prevents using this methodology in open MAS. However, an extension [Sierra et al., 2006] adds an additional specification step (the Social design phase) to define an Electronic Institution with the social structure specified in previous phases. Hence, EI's infrastructure (i.e. Ameli) can be used to ensure that the defined conventions are followed even if agents are designed by third parties. In other words, at the end of the process Ameli provides an organisation infrastructure to support the designed organisation specification.

Analogously, *CArtAgO* [Ricci et al., 2006] offers just one of both components too. Rather than offering a model to specify an organisation, it provides



(a) Organisational structure (b) Organisational Sequence

Figure 2.6: Agent-Group-Role diagrams from [Ferber et al., 2004].

a means to develop an infrastructure to support it. In particular, this approach is based on the Agents & Artifacts meta-model (A&A) [Omicini et al., 2008] which adds a set of tools modelled as objects (Artifacts) which populate agents' environment (Workspace). These objects are not agents, since they do not have individual goals. Instead, they are tools that can be used by MAS participants to pursue their individual goals. Hence, depending on the available artifacts, agents may be able to perform different kinds of actions. This influences the way they interact —i.e. their coordination model. Consequently, given an organisation specification, some of it could be translated into certain artifacts that agents can use in a given workspace. For instance, if an agent a gives the power to agent b to join a group that performs activity c, the former (a) transfers a key-artefact to the latter (b). Such a key-artefact lets the latter (b) open a door-artefact that leads the agent to enter a virtual space –the workspace– where activity c is performed. Precisely, in ORA4MAS [Kitio et al., 2008] this idea is applied to the Moise framework. That is, this work uses Moise to specify an organisation and the artifacts approach to build its infrastructure.

There are more organisational methodologies that are interesting at conceptual level –like OperA [de Almeida Júdice Gamito Dignum, 2004], Soda [Omicini, 2001], Ingenias [Pavón and Gómez-Sanz, 2003] or Gaia [Zambonelli et al., 2003]– but to the best of our knowledge, they do not have a run-time infrastructure to support the designed organisation. In some cases, these methodologies offer an Integrated Development Environment (IDE) to assist in their implementation –e.g. Operetta [Okouya and Dignum, 2008], IngeniasDK [Gómez-Sanz et al., 2008]– and/or use existing infrastructures —e.g. TuCSoN or CArtAgO to support SODA [Molesini et al., 2007], or JADE to support Ingenias [Gómez-Sanz and Pavón, 2005].

Furthermore, *FIPA* has also some standards that correspond to this Organisational Layer. For instance, FIPA-ACL has a set of protocol specifications

to determine message sequences (so-called Interaction Protocols) which are already supported by JADE . Even more, it includes specifications to establish large protocols such the English auction protocol [FIPA, 2001g]. Also, FIPA had some preliminary specifications about normative issues known as Policies and Contracts [FIPA, 2001f], which are surveyed by special agents (so-called Lawyer Agents). In addition, there is an Agent Discovery Service Specification [FIPA, 2001b] which provides the directory service. Overall, different platforms provide these FIPA services, like Thomas [Argente et al., 2008]. It is an organisational infrastructure that extends FIPA Abstract Architecture with transparent interaction among agents and web services. In fact, this infrastructure considers other organisational components not present in FIPA, such as deontic norms.

2.5 Assistance layer

We observe some features in existing works which go beyond enabling a coordination model, assisting agents to participate in it. Accordingly, under our Coordination Support vision we place them in an additional layer called Assistance Layer. The list of services of this layer are mainly conceived by us, as opposed to previous layers, in which services try to encompass existing approaches.

In this related work section, we simply enumerate these services and cite some current approaches that we see as embryonic implementations of this layer. In contrast, a detailed definition of each new service, and its illustration in some of the presented scenarios is given in chapter 4.

2.5.1 Services

We conceive a set of services to assist agents in following a coordination model more than just enabling it. These services may simplify agent development by providing new system facilities that agents can use instead of implementing them individually. However, agents can choose to use these services or not, and its their responsibility to decide which actions to perform. In other words, these services facilitate agent's decision making without reducing participant's autonomy. We call this set of services *Agent Assistance*, which comprises:

- the *Information service* to inform agents about useful information to participate in the MAS.
- the *Justification service* to provide justifications about the consequences of their actions.
- the *Advice service* to suggest alternative plans that conform social conventions.
- the *Estimation service* to estimate the possible consequences of certain actions due to current conventions.

As mentioned above, chapter 4 provides further details and examples about this new services.

Besides, the Assistance Layer may also adapt previous layers depending on system's evolution to improve its performance. In other words, this layer may include pro-active capabilities that let the MAS infrastructure take the initiative and act intelligently. We call these services *Organisational Assistance*, and currently we suggest the service that is the focus of this thesis:

- the *Adaptation service* to update previous Organisational Layer in order to improve system's performance under varying circumstances (see chapter 5 for further details and examples).

This organisational service can be useful when MAS participants behaviour differ notably from the expected one or when the MAS environment changes —like resource availability or technological updates. Such a service is motivated by the computational organisational theory, which claims that the best organisation designs are domain and context dependent [Carley, 1995]. Accordingly, this service could update the organisation when its context changes. Even more, this service can be seen as a reconfiguration aspect of autonomic computing [Kephart and Chess, 2003] in which the MAS as a whole is able to reconfigure itself without human intervention. In other words, under the term Organisational Assistance we consider the system –and its organisation in particular– as an entity that can present different self-* properties [Berns and Ghosh, 2009] as a whole.

In particular, according to the reorganisation typology in [Dignum et al., 2005], we are not talking about a behavioural change but about a structural one. That is to say, instead of referring to a change in agents playing certain roles while the organisation remains the same (so-called a behavioural change), we actually refer to a change in the organisational structures (a structural change). As this typology states, the organisational design acts as an organisational memory, that allows to reflect on the difference between desired and actual behaviour as well as to decide on structural changes.

The following subsections review existing approaches which somehow present some of these Assistance Layer services.

2.5.2 Agent Assistance approaches

Currently, there are some MAS approaches that present mechanisms that can be construed as the first of agent assistance services: the so-called *Information service*. For instance, the Information Services in Thomas [Argente et al., 2008] provide internal agents with information about all the organisational components —i.e. these components are explicit and agents can access their specifications. Similarly, in the A&A approach [Ricci et al., 2006], an artifact provides information about how it can be used (they call it Manual). Additionally, in an Electronic Institution [Esteva, 2003], the staff agent in charge of an interaction activity (Scene

Manager) informs participants when an agent joins/leaves an interaction protocol (a scene). Even more, the mediator among the EI and a participant (Governor) notifies the agent when an action has been filtered out because it did not conform social conventions.

Besides, in Moise [Boissier and Gâteau, 2007], the special agent in charge of the organisation (OrgManager) informs participants when they acquire new obligations. Moreover, agents can ask the OrgManager about which missions are they forced to commit to and which individual goals they can pursue at a certain moment —recall that Moise follows a task assignment approach.

Furthermore, there are some infrastructures that offer a reputation mechanism to agents. We regard such a mechanism as an information service, since the reputation information helps agents to decide how to deal with other participants. For instance, in the Testimony-based Governance Mechanism [Duran et al., 2008], a participant can send a message (Testimony) describing if another agent has or has not violated a social convention. Later, the infrastructure mechanism in charge of collecting such messages (Judgement System) provides a decision (Verdict) about if the agent has actually violated the social convention. These verdicts are accumulated by another infrastructure mechanism (Reputation System) that can be checked by any participant before interacting with other agents. Hence, such a reputation mechanism is also an emergent alternative to organisational enforcement as stated in RepAge [Sabater-Mir, 2006] and Ostracism Enforcement [de Pinninck et al., 2008]. However, we do not include this mechanism as an organisational enforcement policy –in previous layer– because it provides no guarantee —specially in open MAS.

On the other hand, to the best of our knowledge, there are no current approaches that provide neither justifications, advices nor estimations. For example, neither Moise nor EI provide any *justification* when they filter out agent actions. Hence, we suggest these organisation infrastructures could provide justifications in such cases. For instance, in addition to inform that an action has been filtered out, EI's governors could also detail which protocol and which step was not fulfilled. Chapter 4 presents several other examples about these services.

2.5.3 Organisational Assistance approaches

There are several related works on adapting organisations. We consider these approaches provide the *Organisational Adaptation* service proposed within the Organisational Assistance. On the one hand, in ACMAS, these approaches can be divided [Sen and Sen, 2010] into interaction-based emergence (i.e. agents converge in new behaviours by focusing on individual utilities [Mukherjee et al., 2008]) and observation-based adoption (where agents observe others' behaviour [Sen and Airiau, 2007]). On the other hand, in OCMAS, the agents that update the organisation reason about it as an explicit organisational entity, by observing the system as a whole and updating its explicit components.

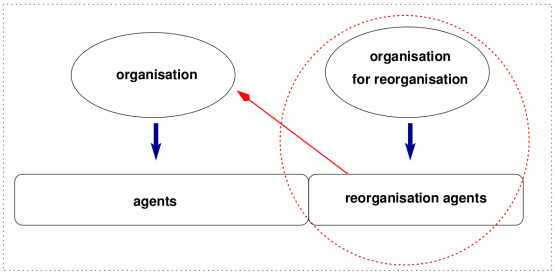


Figure 2.7: Moise's reorganisation architecture from [Hübner et al., 2004].

Next, we describe some AOCMAS, specially regulation definition approaches, since they are the focus of this thesis. In fact, the most of OCMAS approaches presented in previous section are indeed AOCMAS since they also define how to perform the adaptation of their organisational models.

2.5.3.1 Task assignment approaches

In AOCMAS, most of the works are task assignment approaches, so they assume they have full control over the development of MAS participants. Accordingly, they usually derive new tasks to fulfil organisational goals when there are environmental or population changes. Also, they may replicate agents and/or update their social structure to obtain the expected system outcomes.

For instance, in *Moise* they use a new organisation to re-organise the original one as depicted in figure 2.7. In this new organisation, there is a special role (Reorg) that has assigned the task of re-organising other agents. Thus, it decomposes this task and assigns sub-tasks to other reorganisation agents (ReorgExperts) in charge of analysing which changes are required. In particular, in [Hübner et al., 2004] these agents use reinforcement learning to perform such a task. However, different techniques can be applied to perform this task, such as diagnosis [Horling et al., 2001], generalised partial global planning (GPGP) [Lesser et al., 2004] or a knowledge base of organisational structures (KB-ORG) [Sims et al., 2008].

The described Moise's adaptation architecture follows implicitly a meta-level abstraction [Corkill and Lesser, 1983]. In particular, a set of agents is in charge of adapting the organisation of another set of agents. Also, there are other approaches that follow this abstraction. For instance, to the best of our knowledge, the task assignment proposals closer to our approach are: MASPA [Zhang et al., 2009] and Adaptive-MAS Model [Guessoum et al., 2004]. The former, the *Multi-Agent Supervisory Policy Adaptation* (MASPA), has the distributed mechanism depicted in figure 2.8. It is composed by agents that have a partial view of the whole system (so-called Supervisors). Such agents are organised (within a multi-level Supervision Organisation) in order to adapt the organisation of domain agents (Workers) which are grouped into clusters (delimited by dotted lines). These agents provide some kind of norms to agents

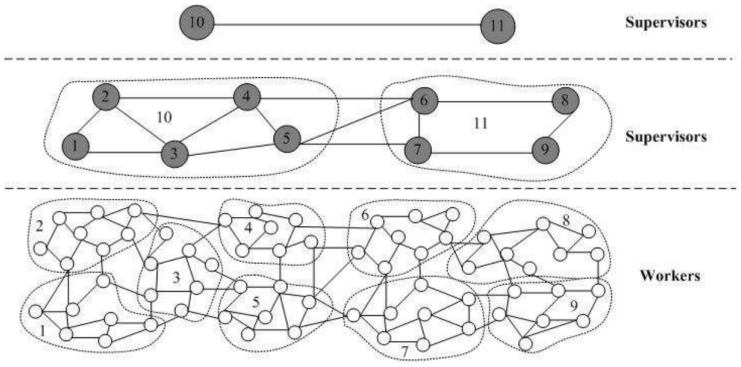


Figure 2.8: MASPA's architecture from [Zhang et al., 2008].

in their previous layer. These norms can be optional local conventions (Suggestions) or mandatory (Rules). In both cases, they specify a condition and some actions. In fact, this approach assumes agents are implemented to check such conditions and perform its corresponding actions. This way, supervision agents integrate global information into the Multi-Agent Reinforcement Learning (MARL [Busoniu et al., 2007]) algorithm executed by its workers. In other words, MASPA aims to create adaptive MAS developing all its components, whereas our mid-term purpose is to deal with open MAS —where agents are developed by third parties, so there is no control over their development and corresponding behaviour.

Similarly, the latter cited meta-level approach, the *Adaptive-MAS Model* [Guessoum et al., 2004], assumes that the adaptation mechanism has total control over domain agents. Specifically, it has a meta-level (so-called Organisation-level) composed by agents (Agent-Monitors) that observe and control the domain agents (Micro-level or Agent-level). Each agent-monitor is in charge of a single agent and sends its information to a collector agent (Host-Monitor). Next, these host-monitors act as a hub of information by exchanging this information to build a global view of the system. This global view may motivate different adaptations like the replication of critical agents that they suggest.

2.5.3.2 Regulation definition approaches

Usually, regulation definition approaches update social conventions, as an indirect tool to vary system's organisation while preserving agent's autonomy. As described in previous layer, these social conventions may include protocols and norms. As an illustration of the former (protocol adaptation), a *Dynamic Argumentation Protocol* [Artikis et al., 2009] is a protocol that can be updated by agents as depicted in figure 2.10. In particular, some of the agents that use a given protocol (so-called Object Protocol) can suggest to change it —such agents are the ones empowered to perform this action. They express this suggestion to

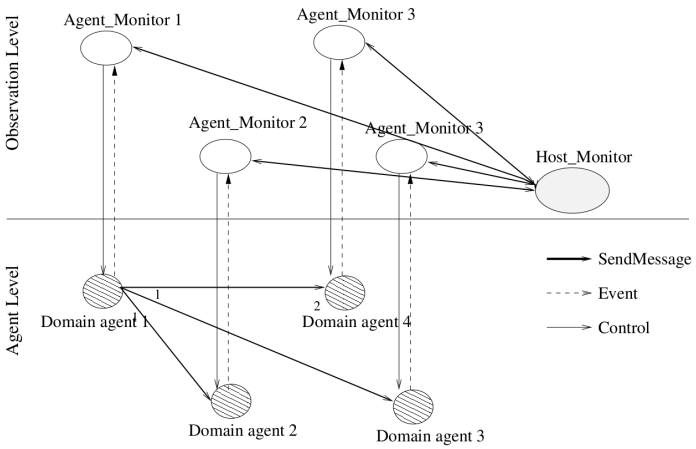


Figure 2.9: Adaptive MAS model's architecture from [Guessoum et al., 2004].

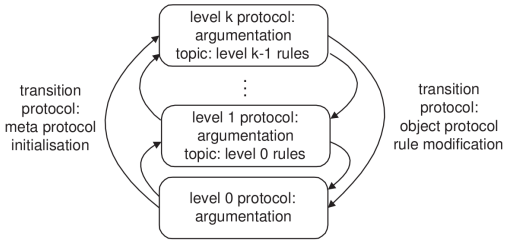


Figure 2.10: Dynamic Argumentation Protocols from [Artikis et al., 2009].

initialise a discussion in a different protocol (Transition Protocol), which is the same they use later to express the actual changes in the object protocol. The discussion itself is performed by using a specific protocol designed for this task (Meta-level Protocol). In other words, some participants can use a meta-level protocol to discuss how to update their regular protocol.

As an illustration about the latter (norm adaptation), the *P2P Normative System* [Grizard et al., 2007] depicted in figure 2.11 is an OCMAS that focuses on the adaptation of two types of norms. On the one hand, there are some norms that are global and mandatory (Rules). Such norms are enforced by restricting interaction, thanks to a reputation service offered by a meta-level (Overlay System). And on the other hand, there are other norms that are local and optional (Conventions). The same meta-level also offers information about their violations, which favours the emergence of groups of agents using similar conventions. In particular, this meta-level consists in an agent (Controller Agent) in charge of each domain-level participant (Applicative agent). Each controller agent observes its applicative agent actions and updates a reputation value depending on its global rules fulfilment. Additionally, a controller cx of an

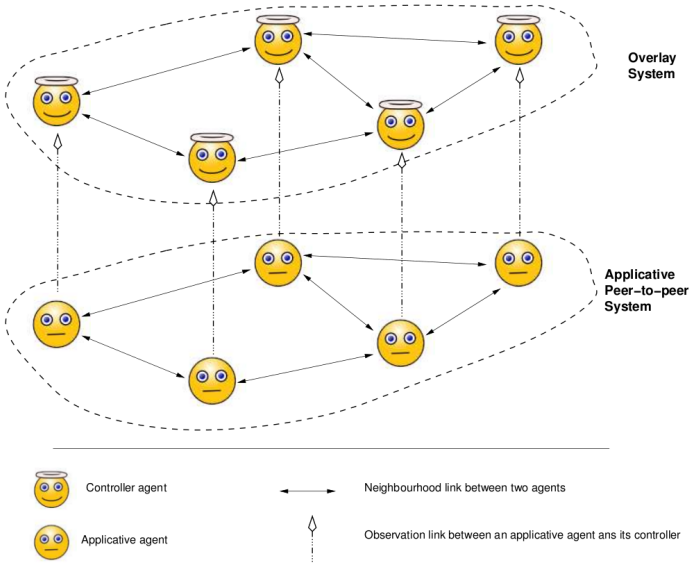


Figure 2.11: P2P Normative System from [Grizard et al., 2007].

agent x, can be informed by another controller cy of agent y, when y is violating any of the local conventions defined by x. This way, agent x can know which of their neighbours is sharing the same local conventions. Overall, in this approach agents can adapt their local conventions but not the global rules. And, the derived social structure depends on norm fulfilling.

Another social conventions adaptation example can be found in an *Autonomic Electronic Institution* (AEI) [Bou, 2009] as depicted in figure 2.12. In this approach, the norms of an electronic institution are adapted when certain system-wide measures differ from the expected ones. In particular a centralised feedback mechanism compares observations (Properties) with their expected values (Goals) and self-reconfigures (its Performative Structure PS and Normative Structure NS, see §3.2) depending on an adaptation policy (Transition Function). The case study of this approach is a traffic scenario, in which there are police agents that can fine cars depending on current norms. These fine amounts are adapted by the feedback mechanism. It is worth mentioning that AEI is the basis for our initial proposal which is explained in chapter 3.

Besides, the mapping between norms and system outcomes may be more complex than the mapping between tasks and goals in a task-oriented problem. In fact, AEI uses machine learning to deal with such complexity. In particular, in [Bou et al., 2007] they apply a genetic algorithm to evaluate goal satisfaction when using different norms (NS). Afterwards, the norms of the best individuals are stored in a Case-Based Reasoning (CBR) [Aam, 1994] system, which is in charge of the adaptation function (Transition Function). Then, during a regular execution, the CBR retrieves similar previous situations (similar Properties, in

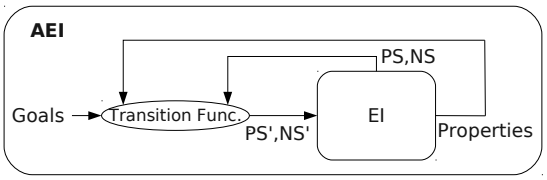


Figure 2.12: Autonomic EI (AEI) adaptation mechanism.

terms of AEI) in order to apply an equivalent solution (a PS', NS' similar to the ones that provided good results in the past). Such a method requires a knowledge-base (called Case-Base by the CBR community) that contains the collection of previous cases (a situation, so-called problem and its solution). According to CBR taxonomy [Plaza and McGinty, 2006], AEI's approach has a centralised processing (a single mechanism to compute the Transition Function) and a centralised knowledge (a single case-base). Alternatively, we propose a distributed CBR approach in our second approach as described in §5.5.

Returning to the ACMAS/OCMAS distinction, there are several works that focus on norm emergence [Pujol et al., 2005, Savarimuthu et al., 2008] and/or norm adaptation [Salazar-Ramirez et al., 2008, Kota et al., 2009] from an *AC-MAS perspective*. However, these approaches use methods that depend on participants' implementation and they rarely create or update organisational persistent entities.

Moreover, further than the cited MAS approaches, there are some works that deal with adaptation in the same scenario that we use as case study. In particular, there are some approaches that deal with a P2P sharing network from a *network management perspective* by reducing communication costs. On the one hand, some of them try to achieve it without ISPs involvement, like Ono [Choffnes and Bustamante, 2008]. In particular, Ono suggests that peers use information collected by Content Distribution Networks (CDN) [Verma, 2002] to select their neighbours. CDNs use dynamic DNS redirection to send clients to low-latency replica servers. Thus, if two clients are sent to the same replica server, they are likely to be close to each other. Accordingly, peers query CDN servers and store the corresponding replica identifiers. Then, when a peer contacts another one, it estimates their distance based on the set of replica identifiers. If it is closer than some of its neighbours (i.e. they have more replica identifiers in common), it starts the sharing process with the other peer. On the other hand, there are approaches that involve ISPs. For example, P4P [Xie et al., 2008] adds an additional layer with elements called iTrackers. These elements can access network information —e.g. topology or traffic measurements. Afterwards, they use it to estimate the distance –in terms of network latency– among peers and suggest different neighbours to a peer depending on this information. As it can be observed, both approaches only adapt part the social structure —i.e. the do not adapt social conventions (protocols or norms) at all.

Chapter 3

Situated Adaptive Electronic Institutions (SAEI)

This chapter defines the Situated Autonomic Electronic Institutions (SAEI) approach. It is based on the organisational model of Electronic Institutions (EI) adapted using the existing Autonomic Electronic Institutions (AEI) approach. Moreover, the resulting approach can provide this adaptation service to an already existing MAS thanks to the Situated Electronic Institution (SEI) model we propose. Overall, the resulting approach (SAEI) is the antecedent of our OCMAS abstract architecture proposal (2-LAMA), which is detailed in a subsequent chapter.

3.1 Introduction

Chapter 2 reviews the state of the art of Adaptive Organisation Centred Multi-Agent Systems (AOCMAS) from our coordination support vision. We assume AOCMAS count on an *Organisational Layer* that provides them with organisation entities in order to structure their coordination model. Moreover, we propose they have an *Assistance Layer* that is able to adapt such organisations to keep their performance under varying circumstances. In particular, we regard this *adaptation* as an *Organisational Assistance*, assuming that organisations are first-class entities that can present self-* properties [Berns and Ghosh, 2009] as a whole. Hence, the state of the art chapter reviews different existing approaches from this perspective. Among others, it cites two approaches that inspire our work: the Electronic Institutions (EI, see §2.4.3) as an organisational approach and the Autonomic Electronic Institution (AEI, see §2.5.3.2) as its adaptation approach.

Now, current chapter defines our Situated Autonomic Electronic Institution (SAEI) proposal using these two approaches plus our suggested Situated Electronic Institutions (SEI). In particular, it uses the AEI centralised adaptation

mechanism extended with the capability to connect to an existing MAS —what we term *situatedness*. This brings two separated agent properties (situatedness and adaptation) to an organisational level, or, in other words, it brings up individual agent level capacities to collective system capabilities. In fact, these capabilities can be added to basic electronic institutions separately. Hence, next subsections present each capability by separate. Afterwards, a SAEI is defined as the composition of these approaches.

The resulting approach (SAEI) is the antecedent of our OCMAS abstract architecture proposal (2-LAMA), which is later detailed in chapter 5. In contrast to SAEI, 2-LAMA has a distributed adaptation mechanism and uses an organisation general model instead of an electronic institution.

3.2 Electronic Institutions applied to our traffic scenario

This subsection is devoted to further introduce Electronic Institutions (EI) –see §2.4.3– since they are referred by subsequent sections. In order to illustrate some of its components, along this section we provide an example based on the traffic scenario described in §1.2.2.1 plus some convenient extensions (e.g. policemen can fine cars).

In brief, the main focus of an Electronic Institution [Esteva, 2003] is to guarantee that its social conventions –interaction protocols and rules– are followed by participant agents, which interact via dialogical actions. This is achieved by communication mediation, so that EIs filter out non-permitted actions — i.e. they have an enforcement policy based on full detection and imposition. Figure 3.1 depicts how MAS regular agents are considered to be external to the institutional framework, and they interact through an institution wrapper called *governor*. Nevertheless, the institution delegates its functions to a special kind of agents, the so called *staff agents*. All in all, the definition of an Electronic Institution is shown below and some of its components are discussed afterwards:

Definition 1 *An Electronic Institution is a tuple* $EI = DF \times DC$ *[Sierra et al., 2007]:*

- $DF = O \times M_I \times ST \times L_{CL} \times L_E$ *stands for Dialogical Framework and provides a context for agent interactions, which are speech acts. Its components are: an Ontology* O*, a set of Information Models* M_I *–to keep information about EI's participants and activities at run time–, a Social structure* ST *–roles and their relationships–, a Communication Language* L_{CL}*, and an Expression Language* L_E *—to specify conditions with a constraint language and their consequences in an action language.*
- $DC = PS \times NS$ *stands for a Deontological Component which is a set of conventions that constrains possible illocutionary exchanges and manages the responsibilities established within the institution. Its components are: a Performative Structure (PS) and a Normative Structure (NS).*

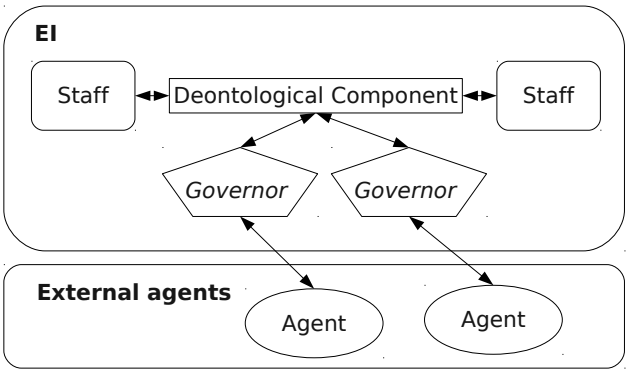


Figure 3.1: Electronic Institution's organisational infrastructure.

3.2.1 Communication Language

The *Communication Language* (L_{CL}) is the language used by agents to utter their messages. Its expressions, called *illocutions* (I), are defined in the terms shown in eq. 3.1.

$$I \quad ::= \quad \iota(origA_i : origR_i, [destA_j :] \; destR_j, msg, t) \tag{3.1}$$

Where there is an illocutionary particle ι (e.g. `request`, `accept`, `inform`...), its sender (origin agent identifier $origA_i$ and the role $origR_i$ it plays), its receivers (destination agent identifier $destA_j$ and/or its role $destR_j$), a message content $msg = f(params)$ and a time stamp[1] t. For instance, in the extension of the traffic scenario –see §1.2.2.1– the following message could appear when police officer 'Bond' informs car 'Shiny' that it has a 10-point fine at time 1: `inform`(`Bond` : `policeman`, `Shiny` : `car`, `fine`(`10`), `1`).

3.2.2 Performative Structure

A performative structure (PS) defines those conventions that regulate the flow of illocutions in an institution. The whole activity of an EI is a composition of multiple, concurrent dialogic activities –the so called scenes– involving different groups of agents playing different roles.

In order to illustrate the PS components, figure 3.2 depicts the PS for the extended traffic scenario. It shows a directed graph, where nodes represent activities (Scenes) and arcs indicate which roles move between them. Thus, in our example, there is an 'Initial' scene where agents enrol in the institution. Afterwards, depending on their role, they can move to another scene: policemen can go to 'Police Station', cars to 'Crossroads' and the rest to 'Collision'. In case there is a collision in this 'Crossroads' scene, crashed car agents move to

[1] EIs have a distributed architecture assuming a synchronised time.

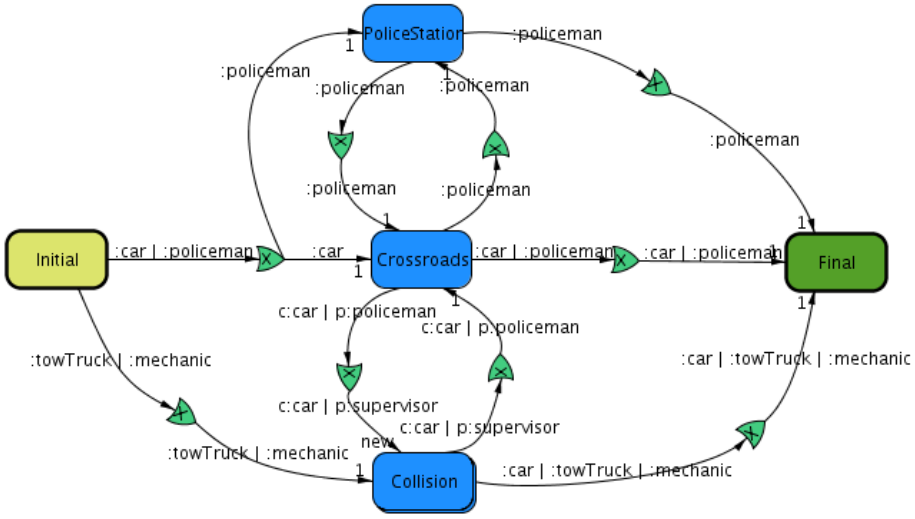


Figure 3.2: EI's Performative Structure in a traffic scenario.

'Collision', which describes a collision emergency protocol (see figure 3.3) that is initially supervised by a police agent.

Each scene is specified by means of a finite-state directed graph, with nodes representing states and arcs defining those relevant actions that imply state transitions. It also includes some restrictions about time variables or how many agents can play a given role. Following the traffic example, figure 3.3 depicts the 'Collision' scene protocol. A crashed car agent enters the scene through an initial state. The scene changes to a 'Garage' state when a 'tow truck' agent informs it has taken this car to a garage, so that it can now –together with the 'supervisor'– exit the scene because a 'mechanic' agent will enter it. Finally, when this mechanic informs the car that has been repaired, the protocol changes to 'Ready' state and both agents can leave the scene.

3.2.3 Normative Structure

The normative structure (NS) [Gaertner et al.,] defines a normative level in our Deontological Component. Both PS and NS are distributed and controlled by staff agents, called Scene Managers and Normative Managers. Briefly, a NS consists of a normative state (S) and a set of rules[2] (SR) that can update this state as shown in eq. 3.2.

[2]It is worth to mention that, in current chapter, we are using the nomenclature of the Electronic Institution's related work on its Normative Structure [García-Camino et al., 2006], which named *rule* –see §3.2.3– to what we will later refer as *norm* in our generic organisation model —see §5.3.2.

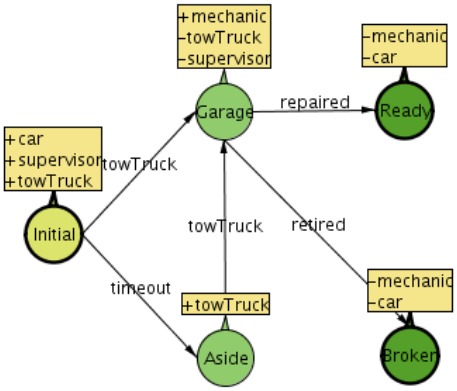


Figure 3.3: "Collision" scene in a traffic scenario.

$$
\begin{aligned}
&NS = S \times SR \\
&S = \{\, P_1 \ldots P_k \,\},\ P_i \in P \\
&\qquad P := \texttt{utt}(I) \| NP \\
&\qquad NP ::= \texttt{per}(I) \| \texttt{prh}(I) \| \texttt{obl}(I) \\
&SR = \{\, R_1 \ldots R_m \,\},\ R_i \in R \\
&\qquad R ::= Cond \Rightarrow Conseq \\
&\qquad Cond ::= \texttt{utt}(I) \| NP \| Cond, Cond \\
&\qquad Conseq ::= \texttt{add}(NP) \| \texttt{remove}(NP) \\
&\tau : S \times R \rightarrow S
\end{aligned}
\tag{3.2}
$$

A *normative state* (S) contains a set of statements called *normative positions* (NP), which represent obligations (`obl`), prohibitions (`prh`) and permissions (`per`) associated to illocutions (I). This state can be updated by agents utterances (`utt`) and *rules* (R). A *rule* [García-Camino et al., 2006] consists of a condition and its consequences. When it is triggered (τ) by any combination of uttered illocutions (`utt`) and NP, it adds or removes NPs to S.

As an illustration, eq. 3.3 contains an example in the traffic scenario. First, it has a social convention (a norm n) about respecting traffic lights. Next, the normative specification (ns) includes a rule r to define the consequences of violating this norm. This rule should say that "any car violating the norm will be fined". However, our example delegates violation judgements to staff agents (policemen in this case). Therefore, the corresponding rule codifies the obligation of a policeman to fine a car when it informs the car went through a red light. Finally, the example shows an execution case. It starts with the empty normative state s_0. Then, when a policeman informs that a car has gone through a red traffic, the original normative state incorporates the corresponding illocution, resulting in s_1. Afterwards, a Normative Manager applies rule r by adding an obligation to normative state s_2.

CONVENTION:
n="cars cannot go through a red light"

SPECIFICATION:
$ns=(s_0, sr)\,, \qquad ns \in NS, s_0 \in S, sr \in SR$
$s_0 = \{\}$
$sr = \{r\}$
r=utt ($inform(x, \texttt{policeman}, y, \texttt{car}, noStop(\texttt{Tlight}), t_i)$)
$\quad \Rightarrow$add(obl ($inform(x, \texttt{policeman}, y, \texttt{car}, fine(\texttt{5}), t_{i+1})$)) (3.3)

EXECUTION:
s_1={utt ($inform(p, \texttt{policeman}, c, \texttt{car}, noStop(tlight_1), t_1)$)}
s_2={utt ($inform(p, \texttt{policeman}, c, \texttt{car}, noStop(tlight_1), t_1)$) ,
obl ($inform(p, \texttt{policeman}, c, \texttt{car}, fine(\texttt{5}), t_2)$) }

3.3 Situated Electronic Institutions (SEI) model

An Electronic Institution is a MAS that provides an interaction mediated environment within the institution itself. We refer to this environment as the *EI inner environment*. Notice that an EI has total control over this EI inner environment thanks to its governors (see §3.2).

Further, a *Situated Electronic Institution* (SEI) is an enhanced EI that is able to interact with a previously existing environment (i.e. with an existing MAS), so that it relaxes total control in favour of interoperability. The existing environment (world) can be any social system (society, organisation or MAS) having individual actions and interactions that are relevant to the institution. These actions and interactions in the world can be illocutions and non-verbal actions. The SEI-world relationship is accomplished by attaching a SEI on top of a world as depicted in figure 3.4. In fact, there is a communication interface (Bridge) between the institution and the world it is situated in. This bridge allows staff agents to access certain world elements as properties. Such properties can be related to modelled agents $AgsP$, institutional issues Org, or environment facts $EnvP$. Also, there are specialised governors (Modellers) that model a world entity as if it was a regular external agent in the EI. In this way, a SEI can perceive world facts and induce changes on it.

We say that a SEI is *situated* in an existing environment because it receives information about the environment, processes it, and induces some changes in the world. We consider a SEI has a model of the world, which maintains –according to external inputs– and updates —translating changes to external environment. It is worth to mention that it is commonly assumed that agents have a partial perception of the world. Similarly to agent level, at organisational level, world may be also partially observable by a SEI. Moreover, SEI's control over this world can be also quite limited, since it can only induce a limited

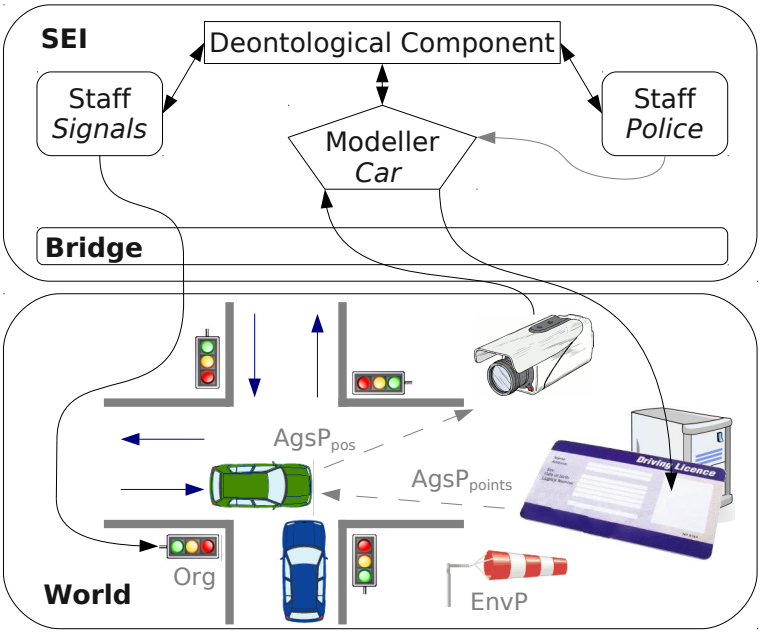


Figure 3.4: Situated Electronic Institution (SEI).

amount of changes in it. Hence, a SEI can be defined as an extension of previous EI definition 1 on page 34:

Definition 2 *We define a Situated Electronic Institution (SEI) as*

$$SEI = DF' \times DC \times B$$

where:

- *DF' stands for a the Dialogical Framework[3] of a regular EI extended with world's entity Modellers and Observable Properties $\subset M_I$.*
- *DC corresponds to the Deontological Component[4] of an EI.*
- *B stands for a Bridge, which is the communication channel with the world.*

Next subsections provide further details about these SEI components.

[3] In brief, the regular DF stands for an agent interaction context that comprises: an ontology O, some information models M_I, a social structure ST, and some communication L_{CL} and expression L_E languages.

[4] In short, DC stands for a set of conventions which comprise: a performative structure PS and a normative structure NS.

3.3.1 Modellers and staff agents

Since a basic EI is a persistent organisation entity, it uses its Information Model (M_I) to keep part of its computational state in the form of attributes. In particular, it has: Agent Observable Properties ($AgsP \subset M_I$), which are those attributes that keep the institutional state of each external agent (e.g. agent's credit or position); Environment Observable Properties ($EnvP \subset M_I$), which are those attributes about global facts independent of the institution activity (e.g. date or weather); And Institutional Observable Properties ($Org \subset M_I$) which are attributes related with global facts directly or indirectly influenced by the institution (e.g. the number of collisions which may be influenced by traffic lights' frequencies).

Some *external agents* of a SEI are represented by relevant world entities that are not controlled by the institution. Thus, a SEI has specialised governors, we name Modellers, in charge of modelling and interacting with these world entities. Thus, a world entity can be treated by the SEI as if it was a regular participant agent.

The information between a SEI and its world flows in two directions. On the one hand, a modeller models a world entity by accessing the world and extracting relevant information about this entity. As a result, a modeller keeps track of its corresponding agent observable properties ($AgsP$) and utters illocutions when its entity performs actions that are relevant to the institution. On the other hand, a modeller translates interactions from SEI into changes in its world entity's Agent observable properties ($AgsP$). Figure 3.4 illustrates this process in the traffic scenario described in §3.2. First, the "Car modeller" gets its car location ($AgsP_{pos}$) by processing the camera information. If this car (c) is entering the road junction through a given lane ($lane_{id}$), the modeller generates the illocution '$\texttt{inform}(\texttt{c} : \texttt{car}, : \texttt{policeman}, \texttt{entryJunction}(\texttt{lane}_{\texttt{id}}), \texttt{t})$'. This illocution informs all policemen in the 'Crossroads' scene that the modelled car has performed the *entryJunction* relevant action. Later, if modeller is asked to decrease car's driving license points ($AgsP_{points}$, see §3.3.2), it will contact the Traffic Regulation Authority to perform this operation.

We see the institution situatedness as an awareness of the world where it is situated. Thus, we consider a SEI is aware of its world in the sense that it models and affects it. However, its world may or may not be aware of this SEI, depending on the domain. Moreover, some domains may present some restrictions on which information can be accessed or updated, so that it determines the level of SEI-world interaction and awareness. For example, in the traffic scenario, if the car's position is retrieved with camera's image processing, this car may probably not be aware of the SEI. In contrast, if the car is equipped with a Global Positioning System sensor device and sends its position to the Traffic Regulation Authority, it may probably be aware of the existence of a surveillance system like the SEI.

On the other hand, there may be some world entities directly controlled by the institution. In this case, instead of a modeller, a SEI has staff agents in charge of these world entities. Figure 3.4 depicts a staff agent called "Signals", which sends information to the world to set a traffic light colour (Org). Staff agents can

also interact with modellers to access to Agent Observable Properties ($AgsP$) or read Environment Observable Properties ($EnvP$, e.g. the wind's direction).

3.3.2 Social Conventions

The term *relevant action* refers to an action –or interaction– in the SEI external environment (world) that affects its institutional model. Consequently, a SEI perceives or induces a relevant action and binds it to the world. Within relevant actions, we distinguish between: allowed actions –those that follow social conventions– and non-allowed actions —the rest of relevant actions.

Moreover, the term *norm* refers to a social convention regarding agents' interaction. Hence, the *allowed actions* are those that comply with norms. Accordingly, we consider that a norm can be violated if agents do not follow its convention, that is, if agents perform *non-allowed actions*. On the other hand, we use the term *rule* to identify an expression that defines the consequences of agents' actions. Hence, a rule defines the consequences of a norm violation.

In a regular EI, most social conventions are specified through protocols so that governors filter out those illocutions not following them (non-allowed actions). In this way, an EI grants no participant can violate these conventions. In contrast, a SEI does not have such control over the world since it cannot prevent world entities from performing actions (or interactions). Thus, when designing a SEI, we have to pay special attention to the fact that it cannot prevent participants from violating norms. Consequently, it is strictly relevant to specify the consequences of violating these conventions with rules added to SEI's Normative Structure —see the example in §3.2.3.

3.3.3 Bridge

The *Bridge* (B) is an asynchronous bi-directional communication channel between our institution and the world (in [Arcos et al., 2007] it was conceived as a channel connected to a multi-agent simulator). This channel is used by staff agents and modellers to obtain information from the external environment and to induce changes in the world as explained previously. It provides access to agent, institutional and environment observable properties.

Basically, this bridge comes from an implementation requirement to bind a SEI and its world. From an implementation perspective, although it is a single module, it may be distributed among different APIs (Application Program Interfaces) to access different programming objects that interact with world elements.

3.4 Autonomic Electronic Institutions (AEI) applied to our traffic scenario

The aim of a regular Electronic Institution is to guarantee that its social conventions are followed by its participant agents. Such conventions have been

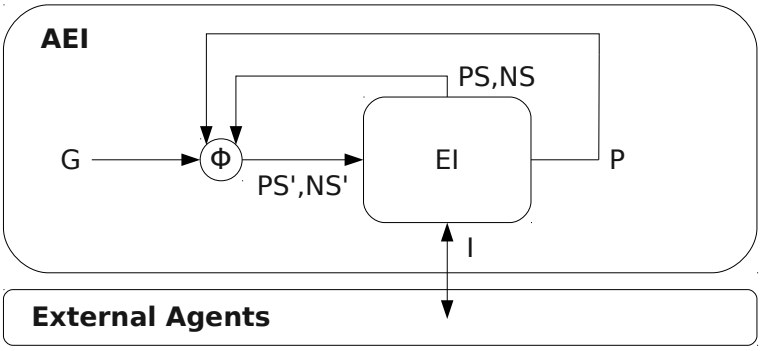


Figure 3.5: Autonomic Electronic Institution (AEI).

designed to pursue some implicit goals. However, as the agent behaviour may differ among different populations, or there may be large environmental changes, the original conventions may not lead to the design goals.

An *Autonomic Electronic Institution* (AEI) [Bou et al., 2006] is an electronic institution that can autonomously adapt itself to achieve a set of defined goals. In fact, goal fulfilment is its driving force for adaptation within the context of a rational world assumption. In this manner, an AEI has an *adaptation* mechanism composed by: a *goal* (G) that specifies desired values of certain properties, the corresponding *observed properties* (P) and a set of *transition functions* (Φ) which define how to reconfigure the institution to accomplish its objective depending on these observations. Figure 3.5 depicts this approach, in which PS and NS stand for the Performative and Normative Structures, and I stands for the illocutions exchanged by external agents through the institution. Accordingly, an AEI can be defined as an extension of previous EI definition 1 on page 34:

Definition 3 *We define an Autonomic Electronic Institution (AEI) as*

$$AEI = DF \times DC \times G \times \Theta$$

where:

- *DF stands for a Dialogical Framework.*
- *DC stands for a Deontological Component.*
- *G stands for the institutional Goals.*
- *Θ stands for the set of Transition Functions which adapt the components of DC.*

Next subsections provide further details about these AEI components.

3.4.1 Institutional Goals

Institutional *Goals* (G) specify desired values for observed properties (P). These properties belong to the information model ($P \subseteq M_I$), and correspond to information about agents, the environment or the institution itself (see §3.3.1). Equations 3.4-3.6 formalise these goals with the components described as follows:

$$G = P' \times \Omega \tag{3.4}$$

$$P' \subseteq P \tag{3.5}$$

$$\Omega = \left[(\mathbb{R})^{|P'|} \rightarrow \mathbb{R} \in [0..1]\right] \tag{3.6}$$

- Relevant Properties (P'): it is the subset of observed properties ($P' \subseteq P$) that is relevant to compute current goals. For instance, in the traffic scenario, if the goals are to keep a low number of norm violations ($p_{\#vio}$) and a minimum number of policemen ($p_{\#police}$), the relevant properties are specified in equation 3.7.

- Objective Function (Ω): it is a function that computes overall goal satisfaction (a real value between 0 and 1, 1 meaning completely satisfied goals) from current values of relevant properties[5]. Notice that it assumes that there is a way to express the value of each property as a real number, so its domain is a real number for each relevant property —i.e. $|P'|$ real numbers. Following the traffic example, the maximum goal satisfaction corresponds to having no violations (i.e. $p_{\#vio} = 0$) and no policemen deployed (i.e. $p_{\#police} = 0$) —which may be an utopian situation, however. Hence, the objective function may be a weighted aggregation function [Bou et al., 2006] like the specified in equation 3.8.

$$\begin{array}{l} p' \text{ is a particular specification of the model } P', \qquad p' \in P' \\ p' = \{p_{\#vio}, p_{\#police}\} \end{array} \tag{3.7}$$

$$\begin{array}{l} \omega \text{ is a particular specification of the model } \Omega, \qquad \omega \in \Omega \\ \omega(x,y) = \frac{1}{weight_{\#vio} \cdot \frac{x}{MAX_{\#vio}} + weight_{\#police} \cdot \frac{y}{MAX_{\#police}}} \\ \quad weight_{\#vio} + weight_{\#police} = 1 \\ \quad x \text{ is a particular runtime value of } p_{\#vio} \\ \quad y \text{ is a particular runtime value of } p_{\#police} \end{array} \tag{3.8}$$

[5] As explained in §5.3, we denote the model of a sort of function as a *function space* ($Model = [Domain \rightarrow Codomain]$, e.g. eq. 3.6) and a specific function of that sort as a *the rule of correspondence of a function* ($spec =< expression >$, e.g. eq. 3.8) that belongs to the function space (i.e. $spec : Domain \rightarrow Codomain$, so $spec \in Model$).

3.4.2 Transition Functions

Transition Functions (Θ) specify how the institution can change its organisational structure with the aim of increasing its overall goal satisfaction. We define two different transition functions depending on what they can adapt (NS or PS). All of them receive a set of observed properties[6] (P) and their desired values (G).

On the one hand, the *Normative Structure adaptation* function (Φ) in charge of updating the rules (NS) is defined in equation 3.9.[7]

$$\Phi : [P \times G \times NS \to NS] \tag{3.9}$$

For instance, in the traffic example, fines could be increased if there are a lot of traffic violations. Hence, ν can adapt propose an updated Normative structure (ns'), by increasing the fine parameter (e.g. from 5 to 10) of rule r (see §3.3) as illustrated in equation 3.10.

$$\begin{array}{lll} ns & = & (s, sr)\,, \qquad ns \in NS, s \in S, sr \in SR \\ & sr & = \{r\} \\ & r & : \texttt{utt}\,(\,inform(x, \texttt{policeman}, y, car, noStop(Tlight), t_i)\,) \\ & & \Rightarrow add\,(\,\texttt{obl}\,(\,inform(x, \texttt{policeman}, y, \texttt{car}, fine(\mathbf{5}), t_{i+1})\,)\,) \\ \phi & (p, & g, ns) = ns', \qquad \phi \in \Phi \\ ns' & = & (s, sr = \{r'\}) \\ & r' & : \texttt{utt}\,(\,inform(x, \texttt{policeman}, y, \texttt{car}, noStop(Tlight), t_i)\,) \\ & & \Rightarrow add\,(\,\texttt{obl}\,(\,inform(x, \texttt{policeman}, y, \texttt{car}, fine(\mathbf{10}), t_{i+1})\,)\,) \end{array} \tag{3.10}$$

On the other hand, the *Performative Structure adaptation* function (Ψ) in charge of updating protocols and/or role flows (PS) is defined in equation 3.11.

$$\Psi : [P \times G \times PS \to PS] \tag{3.11}$$

As an illustration in the traffic scenario, if there are a lot of accidents, function ψ could change the number of allowed policemen deployed in the 'Crossroads' scene —see traffic's PS in figure 3.2.

3.5 Situated Autonomic Electronic Institutions (SAEI) model

It is possible to extend EIs with both situatedness and adaptation capacities simultaneously. The resulting enhanced institution is called *Situated Autonomic*

[6] These properties can be any of the attributes described in §3.3.1.

[7] Notice that this adaptation function (Φ, eq. 3.9) fulfils the general adaptation function defined in next section (Γ^{adapt}, eq. 4.1). Since the observed properties (P) are related to agent and environment properties ($AgsP$, $EnvP$). And, the goals and the normative structure (G, NS) are components of the organisation (Org).

Electronic Institution (SAEI). It can be defined by combining previous definitions 2 and 3:

Definition 4 *We define a Situated Autonomic Electronic Institution (SAEI) as*

$$SAEI = DF' \times DC \times B \times G \times \Theta$$

where:

- *DF' stands for a Dialogical Framework that includes Modellers and Properties.*
- *DC corresponds to a Deontological Component.*
- *B stands for a Bridge (the communication channel with the world).*
- *G stands for institutional Goals.*
- *Θ stands for set of Transition Functions which adapt the components of DC.*

The SAEI model was our first AOCMAS approach. However, it presents some drawbacks: it is specifically based on electronic institution organisation model, it provides a centralised adaptation approach and it is strictly focused on providing an organisational adaptation service. Consequently, in the paper in which we defined it [Campos et al., 2009c], we planned as future work to face these constraints. In fact, we pointed out that it was a step further in order to let existing AEIs deal with such issues: distributing the mechanism and do not require an electronic institution. In particular, we suggested to distribute the adaptation functions (Φ, Ψ) among several staff agents. These staff agents should reason according to their local information, and agree on actual organisational changes. Even more, we also pointed out that there may be other assistance services further the organisational adaptation. Overall, these ideas are considered in our enhanced approach, 2-LAMA, described in chapter 5.

Discussion

Most of literature interprets situatedness at organisation level as providing a location notion to MAS participants. This idea was introduced by Weyns et. al. [Weyns and Holvoet, 2004] as a way to allow local synchronisation of agents in the first *Situated MAS* approach [Ferber and Muller, 1995]. The key point is to restrict participants' perceptions depending on their virtual location. CArtAgO [Ricci et al., 2006] is also an example of this perception paradigm. It provides direct interaction among agents, and also indirect interaction through *artifacts*. But, in both cases, the scope of these interactions is limited to workspaces where these elements are located. It uses an agent body to situate an agent inside those workspaces; then its location determines which artifacts can be perceived or manipulated by its corresponding agent. In this sense, Electronic Institution scenes can be regarded as a way of grouping agents that can

interact together, which can be interpreted as a virtual location that restricts their perception. *EASI* model [Saunier et al., 2006] goes a step further, and additionally lets agents determine which element they want to perceive. They sustain this approach exposing that awareness is an active state. Precisely, we see EI's situatedness as an awareness of the world where it is situated. A SEI, as a whole, determines its world perception and interaction. A first approach to this EI's situatedness was the Simulator Bridge [Arcos et al., 2007]. However, we go further by assuming all external agents' interactions are performed in the world. A similar approach is detailed in [Valckenaers et al., 2007], where they explore the idea of controlling physical entities with a MAS. They perform a global overview, without detailing changes in EIs, but provide additional ideas like augmentation –providing extra information– of real world elements to MAS agents. Our notion of situatedness could be used to provide the same normative environment to different existing systems –updating the Bridge–, like in [Cardoso et al., 2009] where they consider using the same Normative Structure in different Contexts. However, they study the adaptation of such normative context depending on individual goals, while we perform it according to institutional goals. These institutional goals could be initially agreed by participants as suggested in [Gaertner et al., 2009], so they will have a connection with the individual goals.

Chapter 4

Assistance Layer

In this thesis we consider a set of services devoted to provide assistance to MAS participants further than just allowing them to enact a certain coordination model. As stated when reviewing the state of the art, we locate these services in the top coordination support layer, so-called *Assistance Layer*. In this chapter, we further describe these services within two groups related to their scope: collective (Organisational Assistance) or individual (Agent Assistance). In all cases, in addition to the basic examples presented in the related work –see §2.5– we provide more examples in some of our regulation-oriented scenarios. Overall, the organisational adaptation service is the most relevant to the focus of this thesis. Hence, in despite of presenting this service in this chapter, next chapter is entirely committed to formalise it and to propose our approach to provide it.

4.1 Introduction

We observed some features in existing works, that we construe they go beyond enabling a coordination model, but they assist agents to participate in it. Accordingly, we propose to add an additional coordination support layer –see §2.1– called *Assistance Layer*. Such a layer assists agent coordination –rather than enabling it– and comprises new system facilities that alleviate agent implementation. That is to say, this layer aids agents to use more effectively and efficiently the coordination mechanisms provided by previous layers. Thus, we regard it as a step forward in MAS development, that facilitates the engineering and enrolment of heterogeneous agents —this is specially relevant in open MAS since agents are designed by different parties.

This layer provides two main types of services: adapting MAS organisation to varying circumstances (*Organisational Assistance*) and assisting individual agents to achieve their goals under current context (*Agent Assistance*). The former, the *Organisational Assistance*, consists in adapting the existing organisation to improve the system performance under varying circumstances. This can be the case when participants behaviour differs notably from expected one or

when MAS environment changes —such as resource availability or technological updates. That is to say, this layer may include pro-active capabilities that let it take the initiative and act intelligently. Within a rational world assumption, we propose adaptation to be driven by the goal fulfilment criteria. As an illustration, in our traffic example, the speed limit norm can be updated to balance the trip time average and the number of collisions depending on current traffic flow.

The latter, the *Agent Assistance*, may simplify agent development by providing new system facilities that agents can use instead of implementing them individually. However, agents can choose to use these services or not, and it is their responsibility to decide which actions to perform. In other words, these services facilitate agent's decision making without reducing participant's autonomy. Such services comprise providing agents with useful information to participate in the MAS (*Information service*); providing justifications of the consequences of their actions, for example, when an agent action is not allowed (*Justification service*); suggesting alternative plans that conform social conventions (*Advice service*) and estimating the possible consequences that certain actions would have due to current conventions (*Estimation service*). For instance, in the traffic example, an information service can notify cars about updates in the speed limit norm. Afterwards, if a police agent fines a car for exceeding this speed limit, a justification service can detail the violated norm and the detection circumstances. Additionally, an advice service may provide alternative routes, whose trip time can be approximated by an estimation service.

Next sections derive from our initial Assistance Layer description published in [Campos et al., 2009a, Campos et al., 2009b] plus several enhancements related to its description and formalisation.

4.2 Organisational Assistance

We call *Organisational Assistance* to those assistance services that stress the organisational goals provided by previous layer. Hence, we assume MAS organisation is an entity that can present different self-* properties [Berns and Ghosh, 2009] as a whole. Currently, we propose an adaptation service which is mainly related to self-adapting an existing organisation. However, other services related to self-* properties could be explored, such as self-organising (i.e. creating an organisation instead of adapting an existing one).

Next subsection presents the mentioned service, so-called the *Adaptation service*. Furthermore, due to its relevance to the focus of our work, in addition to present this service in this chapter, the rest of the thesis is devoted to formalise it, to propose an approach to provide it to an OCMAS, and to evaluate such approach empirically.

4.2.1 Adaptation service

A MAS organisation establishes a coordination model that help to achieve its organisational goals. However, there can be some changes that bring a new

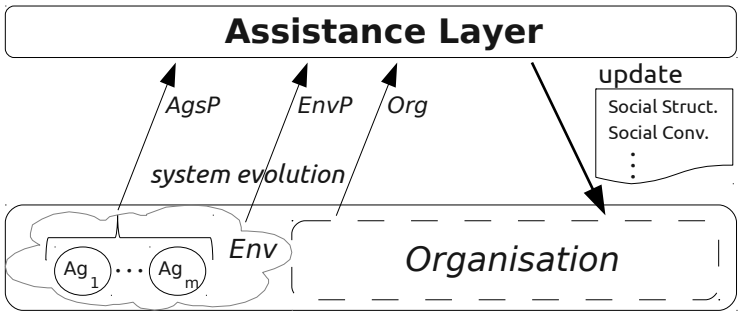


Figure 4.1: Adaptation service.

situation in which the current organisational structures are not accomplishing its original target. For instance, when participant agents' behaviour differs notably from the expected and/or there are significant environment changes —like changes in country laws, resource availability, or technologies. In such cases, the organisation should be adapted to bring the system towards its original goals.

Therefore, we propose that the Assistance Layer provides a pro-active *Adaptation Service* at organisational level as depicted in figure 4.1. In particular, we propose goal fulfilment –in efficacy and efficiency– as the driving force for adaptation within the context of a rational world assumption. Hence, the Assistance Layer requires some way (1) to observe *system evolution*, (2) to compare it with the *organisational goals* and (3) to *update* the organisation in order to improve goal fulfilment. In sum, this service can be formalised as the function (Γ^{adapt}) defined in equation 4.1, where $AgsP$ stands for the participant observable properties, $EnvP$ stands for the environment observable properties and Org stands for the organisation —i.e. its specification and its run-time status.

$$\Gamma^{adapt} = [AgsP \times EnvP \times Org \rightarrow Org] \tag{4.1}$$

Notice that, for the sake of simplicity, in subsequent chapters we also refer to this function Γ^{adapt} as α^O (α stands for adaptation function, and O stands for organisational level) since we will refer to the adaptation of different organisation subcomponents —see equation 5.63.

As an illustration, we consider that the *auction scenario* organisational goal is to maximise the number of transactions and the total amount of transferred money. Also, we consider there is an original social convention that establishes the starting price of an English auction —it is fixed depending on the expected money agents can spend. Then, we assume that along time agent's purchasing power may decrease, so the number of transactions may also decrease due to the initial price fixed by the social convention. Consequently, the amount of transferred money may decay accordingly. In such a case, the Assistance Layer may identify that system performance is decreasing because the transfer parameters are lower than their previous values. Hence, this service may adapt the mentioned social convention by decreasing the starting price. This adaptation

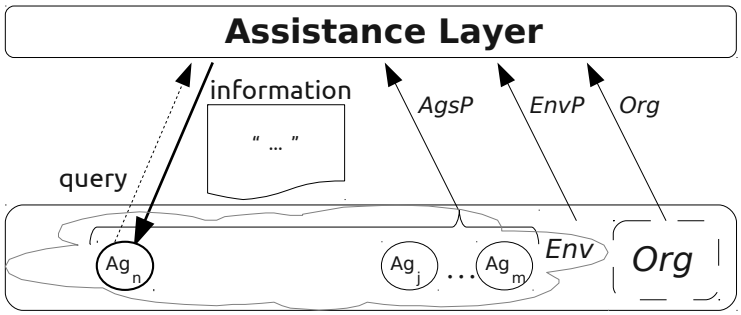


Figure 4.2: Information service.

may hopefully lead to an increment of transfers and the total transferred money. That is to say, this organisational adaptation may let the system fulfil its original goals —see §1.2.2-7.2-7.3 for other adaptation examples.

See next chapter for further details and an entire formalisation of the described adaptation service in regulation-oriented problem scenarios.

4.3 Agent Assistance

We call *Agent Assistance services* to those services devoted to assist agents to achieve their individual goals under current organisation and context. In brief, these services consist in providing information about the system, justifying the consequences of agent actions, giving advices to observe current conventions and estimating what would happen if the agent performs a certain action. Next, each service is specified in a subsection that contains further details about it.

4.3.1 Information service

The base Assistance Layer function is to provide agents with useful *information* about the coordination model and its state which makes them easier to participate in the MAS. Moreover, it can provide participants with information about other agents, like their reputation for following social conventions[1]. In short, this is depicted in figure 4.2, where the Assistance Layer provides an agent (Ag_n) information about participants (Ag_i), their environment (Env) and their organisation (Org). Any of this information may be provided either because it is new for the agents –or has been updated–, because they asked for it –or subscribed to it–, or just as a reminder to emphasise it. This way, an agent does not need to periodically check for updates on certain information, or even store it. For instance, the system can inform a newcomer agent about social conventions and notify other agents about a new participant. Also, any

[1] Notice that MAS infrastructure could build this reputation by observing agents or through opinions submitted by participants (see §2.5.2).

agent can ask the system about the conventions that are applicable in its actual context. Moreover, the system may inform agents when these social conventions are updated. Finally, the system may send a reminder to participants when a scheduled activity is going to start.

In order to formalise these mentioned concepts, we refer to the information about participants as their observable properties ($AgsP$) and the information about the environment as its observable properties ($EnvP$). Besides, the information about the organisation (Org) comprises its specification and its run-time status. Hence, this service can be expressed by the function (Γ^{info}) defined in equation 4.2, where $QueryInfo$ stands for the query message that agent Ag_n may send[2] to the Assistance Layer and $Info$ stands for the resulting information message that this layer sends to the agent.

$$\Gamma^{info} = \Big[(QueryInfo)^{\{0,1\}} \times AgsP \times EnvP \times Org \rightarrow Info\Big] \tag{4.2}$$

In order to illustrate this service in the *auction scenario*, we consider a buyer agent can purchase goods by participating in several auctions. Accordingly, entering agents would receive information about current auctions and their conventions. If they express in which products they are interested in, the Assistance Layer will notify them whenever any of these products is going to be auctioned. Even more, the system can send them reminders when scheduled auctions begin. Also, a reputation service can store the satisfaction level of previous buyers with the products offered by each seller. In such a case, participants may query this service to decide how much they want to bid for a product of a certain seller.

4.3.2 Justification service

Further than informing participants about MAS state and conventions, we propose a service that provides them with a justification about state evolution depending on their actions. In particular, we conceive such a *justification* as the description of facts (cause) that had as a result a certain situation (consequence). So, the Assistance Layer can provide an agent with a justification about the consequence of its actions in their current context —actions and context constitute the cause. Figure 4.3 depicts this process: when an agent (Ag_n) performs a given action that has some effects, the Assistance Layer provides a justification. The performed action plus the current system state and conventions conforms the *cause*, whereas its effects and the subsequent system status conforms the *consequence*. Even more, this layer may also provide a justification to agent Ag_n, when it is affected by a state change triggered by any other factor —i.e. Ag_n may have not performed any action. For instance, depending on the

[2]Notice that we use the superscript $\{0,1\}$ to denote that a certain element is optional. For example, in eq. 4.2, $(QueryInfo)^{\{0,1\}}$ means that the $QueryInfo$ may be present (when an agent queries for information) or not (when the service is pro-active and informs an agent without receiving any query).

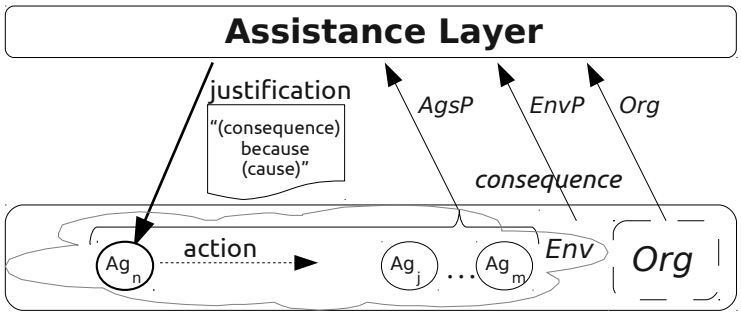


Figure 4.3: Justification service.

enforcement policy, an action can be filtered out or performed with extra consequences —e.g. new obligations or prohibitions. In both cases we suggest to send a justification to the participant explaining the cause of the actual consequence. Overall, notice that in the example, the previous coordination support layer (Organisational Layer) applies the enforcement policy –i.e. to enable– whereas the Assistance Layer explains it —i.e. to assist.

This service can be formalised as the function (Γ^{justif}) specified in equation 4.3, where *Action* stands for the action that agent Ag_n performed and $Justif$ for the resulting justification message that the Assistance Layer sends to the agent.

$$\begin{aligned} \Gamma^{justif} = \Big[(Action)^{\{0,1\}} \times AgsP \times EnvP \times Org \rightarrow Justif \Big] \\ Justif = Cause \times Consequence \end{aligned} \tag{4.3}$$

As an illustration, following the *auction example*, when a bid is not accepted, the bidder agent receives the corresponding justification. For example, in case the amount is lower than the previous bid, the agent will be informed that there is a convention that prevents it. Alternatively, if an agent is fined because its bid is greater than its authorised credit, the justification may indicate this agent that this exceeding was the cause.

4.3.3 Advice service

In addition to justify action consequences, the Assistance Layer could provide an advice to an agent in order to help it to decide which actions it performs. We conceive such an *advice* as a set of alternate plans that fulfil current social conventions. Each plan is a sequence of actions that takes into account current system status as depicted in figure 4.4. Hence, an agent may consider only some restricted sets of sequential actions in compliance with social conventions, instead of facing all possible action sequences. This lets abstracting agents from low level action planning, so they can plan at a higher level. Therefore, this

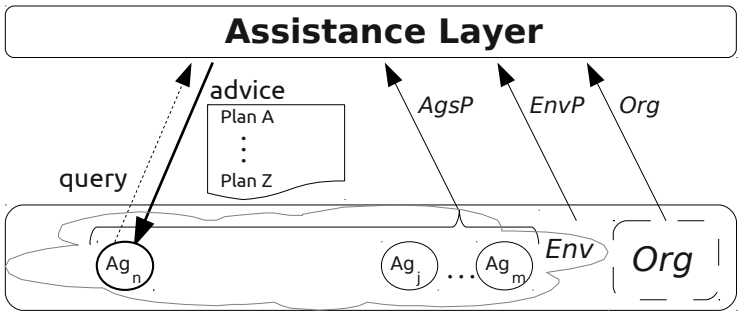


Figure 4.4: Advice service.

may reduce the complexity of developing agents of a MAS that provides this functionality.

The Assistance Layer may provide this functionality in a reactive or proactive way. In the former case, it will only give advices to agents that request them. Whereas in the latter case, it will give an advice when it assumes an agent may need it. For instance, when an agent enrols in a certain activity, or an agent has been inactive for a long time, or when after filtering out an action.

This service can be formalised as the function (Γ^{advice}) defined in equation 4.4, where *QueryAdvice* stands for the query message that agent Ag_n sends to the Assistance Layer and *Advice* for the advice message that this layer sends to the agent —this message contains a set of plans, $(Plan)^*$.

$$\Gamma^{advice} = \Big[(QueryAdvice)^{\{0,1\}} \times AgsP \times EnvP \times Org \rightarrow Advice\Big] \qquad Advice = (Plan)^* \tag{4.4}$$

The way to create an advice can vary from indicating what other agents have performed on the same situation, until to plan possible actions given some restrictions and goals. Restrictions include current social conventions, the context status and agent capabilities. The goals to take into account may be a compound of individual and organisational goals. On the one hand, individual goals can be estimated from an agent's role, its action history or even the individual goals it may have revealed. Moreover, the Assistance Layer can play as a third party when having information of different individual goals, and it can try to coordinate the generated plans that satisfy as many goals as possible. On the other hand, the organisational goals may be provided by previous Organisational Layer, so this service can take them into account. All these goal sources may generate different alternative plans to an agent. Then, this agent may evaluate the plans and follow one of them.

As an illustration, in the *auction scenario*, an entering agent can indicate the product it is searching for and ask for advice. Then, the Assistance Layer may suggest that this agent goes to those English auctions which offer such product with a starting price below agent's authorised credit. Later on, if agent's bid is

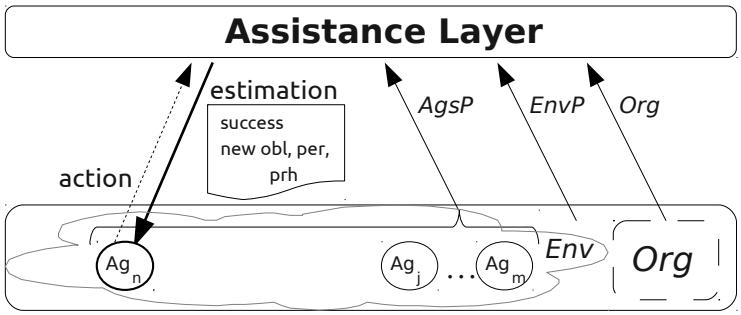


Figure 4.5: Estimation service.

greater than its authorised credit, the Assistance Layer may advise the agent to increase its credit or change to another auction.

4.3.4 Estimation service

Considering that the aim of the Assistance Layer is to assist MAS participants, it could help an agent by offering it an estimation of the consequences of performing a certain action —notice that, in such a case, the action is not performed. As depicted in figure 4.5, an agent (Ag_n) can ask the Assistance Layer to estimate the consequences of a certain action before deciding whether to perform it or not. For example, the Assistance Layer can check whether such an action fulfils current social conventions or not. As a result, it can indicate if the action would be filtered out, or if the agent would acquire new obligations. In this example, the estimation is similar to providing information after an action is actually performed, but without executing it. In addition, an estimation could even include a justification –e.g. indicating which convention would be violated– or even an advice —e.g. suggesting alternative plans depending on estimated consequences. It is worth mentioning that we use the word estimation since the Assistance Layer cannot really know what other agents will do and how the context can change before the agent actually performs the action. Thus, in all cases, the prediction is just an estimation because the action is not really performed, and therefore its consequences and the corresponding justification and advice would be different if it was really executed.

Such a service can be formalised as the function (Γ^{estim}) specified in equation 4.5, where *Action* stands for the potential action that agent Ag_n sends to the Assistance Layer and *Estim* for the estimation message that this layer sends to the agent.

$$\Gamma^{estim} = [Action \times AgsP \times EnvP \times Org \rightarrow Estim] \tag{4.5}$$

We argue that this service also simplifies agent development since the Assistance Layer provides the evaluation of social conventions. Thus, it works as a decision support system for agents, which can directly focus on evaluating action

consequences instead of evaluating conventions. Hence, agent complexity can be reduced because some of its evaluation capacity is delegated to the MAS infrastructure. For instance, an agent could know which obligations would acquire without reasoning about current rules. Moreover, an agent could use a learning mechanism to acquire empirical knowledge about social conventions instead of analysing their specifications. Or simply, a developer could use this functionality to test a new agent in a real situation without any danger of improper action consequences.

As an illustration, in the *auction scenario* a newcomer agent may ask for an estimation in case it bids for a certain product. As a response, it may receive, for instance, an estimation with all possible components (information, justification and advice). The information part may contain current reputation of the product's seller, and the last price paid for a similar product. And the justification part may indicate that the bid would not be accepted since its amount is greater than agent's authorised credit. Finally, the advice part may suggest that the agent can either contact the banking agent to increase its credit or change to another auction where a similar product will be auctioned..

Chapter 5

Organisational Adaptation

This chapter goes further in detailing the Organisational Adaptation service presented in previous chapter and defines our approach to provide it. This approach tries to generalise our first proposal, the Situated Autonomic Electronic Institution (SAEI), which was centralised, based on electronic institutions and strictly focused on providing the adaptation service. Consequently, this chapter starts by formalising a generic MAS organisation and identifying the features present in agents that can perform an organisational adaptation in order to distribute the adaptation mechanism. Then, the chapter presents our enhanced generic distributed approach, so-called Two Level Assisted MAS Architecture (2-LAMA).

5.1 Introduction

After defining Assistance Layer services in previous chapter, current chapter defines our proposal to provide them, mainly the organisational adaptation service. In fact, chapter 3 already presented our first approach to provide this adaptation service: the Situated Autonomic Electronic Institution (SAEI). However, that proposal had the drawbacks of being a centralised mechanism and being tied to a particular organisation model, the electronic institution.

Alternatively, our enhanced proposal, the Two Level Assisted MAS Architecture (2-LAMA), has a distributed adaptation mechanism that relies on a general organisational model. On the one hand, its organisational general model was derived from the state of the art in order to facilitate that our approach could be easily tailored to other organisational models. On the other hand, its distributed architecture is related to the features present in agents that adapt an organisation. Moreover, this architecture is based on the premise of not making assumptions about participants characteristics. This way, 2-LAMA is able to work with our target problems: the regulation-oriented problems defined in §1.2.1. In fact, avoiding participant-related assumptions helps to prevent design decisions that could compromise the application of our architecture in an

open MAS context. As a result, our architecture follows the meta-level abstraction [Corkill and Lesser, 1983] in the sense that the organisation is updated by a distributed mechanism that reasons at a higher level of abstraction and is not involved in the domain activity —as introduced in figure 1.5.

Therefore, this chapter is structured in three sections: one that defines the general organisational model and the notation used in the rest of this document; another one that lists the characteristics of adapting agents; and a last section that accurately defines 2-LAMA. Along these sections, we provide several specification examples in the traffic scenario introduced in §1.2.2.2. In addition, next chapter provides further examples in the P2P sharing network case study introduced in §1.2.2.3. Such case study is also used when discussing about 2-LAMA adaptation alternatives in chapter 7. Even more, afterwards, the 2-LAMA proposal is evaluated in this same scenario, too.

5.2 Notation

This section introduces the notation used to formalise our approach in the subsequent sections. Such notation is based on the mathematical notation used in set theory [Sharma, 2010]. In brief, these notations are used to describe sets and relations among them. We use it to express the components and processes of a computational system in the following three stages: Model, Specification and Execution State —see §2.4.3. Figure 5.1 depicts these stages and its notation on an elementary program example.

5.2.1 Model

First, when defining a *Model*, we mainly use *upper-case letters* to describe either:

- (i) the model of a component *composed by subcomponents*: it is denoted by a Latin-alphabet word and defined as a product set of its subcomponents. For instance, $Sys = Categories \times \Gamma$ means that the systems that fulfils the model Sys, have two subcomponents ($Categories$ and Γ).

- (ii) the model of a *process* with a certain input and output: it is generally denoted with a Greek-alphabet[1] single letter and defined as a function space[2] among the sets of possible process inputs (i.e. its domain) and the set of possible outputs (i.e. its codomain). For instance, equation 5.1 means that in our example there is a process that given two natural numbers provides a certain value that belongs to the $Categories$ set.

[1] In order to avoid misunderstandings, when a Greek upper-case letter is equal to the Latin equivalent, we use a larger version of the lower-case letter —e.g. we use 'α' instead of 'A' as the 'α' upper-case.

[2] In mathematics, a *function* is a relation that associates each element in the domain with exactly one element in the codomain ($function : Domain \rightarrow Codomain$). And a *function space* is a set of functions that share the same domain and codomain ($function\,space = [Domain \rightarrow Codomain]$, e.g. eq. 5.1). This way, by using function spaces, the notation can refer to all the processes that fulfil a given input/output model.

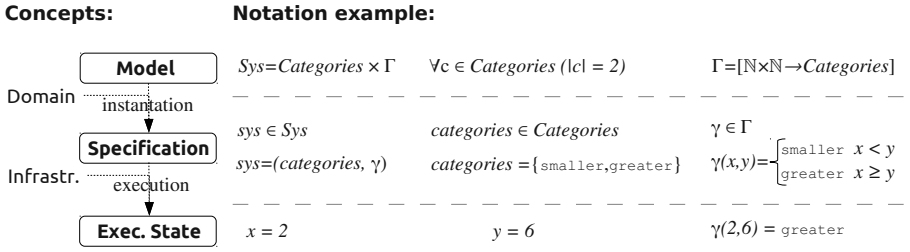


Figure 5.1: Notation example for Model, Specification and Execution State.

$$\Gamma = [\mathbb{N} \times \mathbb{N} \to Categories] \tag{5.1}$$

We can provide further details about any components using standard mathematical set notation. For example, $\forall c \in Categories\ (|c| = 2)$ means that the set of possible categories have two elements. Notice, that in this example, the model is not specifying which are these elements. It just expresses that the systems that fulfil such model have only two possible categories.

5.2.2 Specification

Next, when defining a particular *Specification* that fulfils a certain Model, we generally use *lower-case letters* to describe either:

- (i) a specific *component* according to the given model: it is generally denoted with a Latin-alphabet single letter and defined as an element that belongs to the set described by the model. For example, $sys \in Sys$ means that the specific system sys fulfils the model Sys. Hence, sys has two subcomponents $sys = (categories, \gamma)$ that fulfil the sort of components defined by the model —i.e. $categories \in Categories$ and $\gamma \in \Gamma$.

- (ii) a specific *process* with an input and output according to the model: it is generally denoted with a Greek-alphabet single letter and defined as the rule of correspondence of a function[3] within the function space defined by the given model. For example, equation 5.2 expresses the rule of correspondence of a process that fulfils the domain and codomain described by the model –i.e. $\gamma \in \Gamma$– assuming that x, y are natural numbers.

$$\gamma(x, y) = \begin{cases} \texttt{smaller} & x < y \\ \texttt{greater} & x \geq y \end{cases} \tag{5.2}$$

Again, we can provide further details about any component using standard mathematical set notation. For instance, we may specify the particular elements of a set as constants. So, $categories = \{\texttt{smaller}, \texttt{greater}\}$ means that the categories set has exactly the two specified symbols. Hence, it fulfils the model

[3] In particular, we assume a specific process is a computable function, which means that it can be defined by an algorithm.

restriction since $|categories| = 2$. Notice, that we use a typewriter font to denote constant symbols, so this font highlights that these symbols have no further descriptions nor alternate values —i.e. they are "literals" from a generative grammar perspective [Chomsky, 1965].

It is worth to mention, that a Specification may contain upper-case letters if it needs to express a set of values or a process that will be designated later during the execution. Analogously, a Model description may contain lower-case letters if it already defines or refers to a particular set of values or process forced by the model.

5.2.3 Execution State

Finally, when expressing part of the *Execution State* of a given Specification in a certain moment, we determine the symbols that were undefined by previous stages. As a result, the value of all symbols can be resolved. Notice that such values may change over the time, so symbols should have an additional t subscript or parameter to denote a particular time. However, for the sake of simplicity, we generally omit it and provide their values for a given time. For instance, the symbols that remain undefined in the example are x and y —i.e. they are variables that will have a value at run-time. So, as an illustration, at a given time we assume that $x = 2$ since it was entered by the user and $y = 6$ because it was obtained from a sensor. Consequently, we can resolve that at that given time, $\gamma(2, 6) = \texttt{greater}$.

5.3 General Organisation Model

In order to propose a new abstract AOCMAS approach more general than SAEI, we reviewed the state of the art of organisation models. In particular, we identified the components they usually present in such models in §2.4.2. Figure 5.2 illustrates such components for an organisation (Org) that regulates the activities of a set of agents ($Ags = \{ag_1, \ldots, ag_{\#ag}\}$, where $\#ag = |Ags|$) within an environment (Env). Notice that we assume participant agents do not belong to the organisation itself, so this organisation general model can also encompass open MAS —where the organisation entity persists regardless entering/leaving agents and their behaviour. Moreover, along the description of our approach, we focus specifically on their observable properties ($AgsP$)[4] instead of the agents themselves (Ags). Analogously, we usually refer to the observable properties of the environment ($EnvP$) instead of the environment itself (Env).

Specifically, we define an *organisation model* with the aim to encompass existing approaches, as follows.

Definition 5 *We define the Organisation Model (Org) as*

$$Org = SocStr \times SocConv \times Goals$$

[4]Later, we will use $AgsP^S$ to highlight that it refers to the properties of all agents of the system. So, we will use the S superscript to denote the whole system.

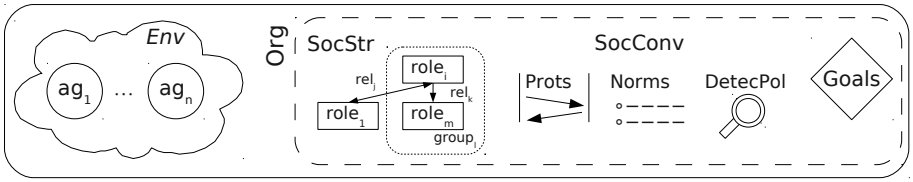


Figure 5.2: The General Organisation Model of an OCMAS.

where:

- *SocStr stands for a Social Structure.*
- *SocConv stands for the Social Conventions.*
- *Goals stands for a set of Organisational Goals.*

Such organisation model has a social structure (*SocStr*) that defines agent roles, groups and relationships. Based on these definitions, other organisational components can refer to participant agents in a generic manner. For instance, the social conventions (*SocConv*) include protocols and norms that restrict the behaviour of specific roles. Such social conventions also include a detection policy that describes how to detect convention violations. In addition, we assume the organisation has explicit goals (*Goals*) that describe its design purpose. Next subsections provide further details about each organisational component.

5.3.1 The Social Structure

A social structure (*SocStr*) contains a set of roles (*Roles*), groups (*Groups*) and relationships (*Rels*) among the agents that play these roles or belong to these groups —see equations 5.3-5.6. In brief, a role specification defines the set of allowed actions for an agent; a group specification describes how agents may team up, and the role relations define which roles can be played simultaneously or what is their authority relation —see §2.4.2.

$$SocStr = Roles \times Groups \times Rels \tag{5.3}$$

$$Roles = \{r_1, \ldots, r_{\#r} : r_i \in Role\} \tag{5.4}$$

$$Groups = \{g_1, \ldots, g_{\#g} : g_i \in Group\} \tag{5.5}$$

$$Rels = \{rel_1, \ldots, rel_{\#r} : rel_i \in Rel\} \tag{5.6}$$

The execution state of these components keeps the binding among these elements. For instance, there may be a function (Π^{Role}) that returns true if an agent ($ag \in Ag$) has a given role ($role \in Role$): $\Pi^{Role} = [Ag \times Role] \rightarrow \{\texttt{true}, \texttt{false}\}$. Moreover, as we frequently use these sort of functions[5] when

[5]Notice that the model of a relation (*Rel*) is the definition of an additional function among agents $Rel = [(Ag)^* \rightarrow \{\texttt{true}, \texttt{false}\}]$, which can be used when specifying other components.

specifying other components (e.g. specifying a norm, like in eq. 5.20), we actually use a shorter equivalent expression like: $\mathtt{R}_{ag_i}$ refers to the set of roles played by a given agent, i.e. $\mathtt{R}_{ag_i} = \{r_1 \dots r_k : r_i \in Role \wedge \pi^{role}(ag_i, r_i) = \mathtt{true}\}$.

As an illustration of these components, the equations 5.7-5.9 detail them in the traffic scenario. For instance, there may be one role for the car agents (`car`) —the police agents are introduced in §5.5.1.4 since we assume they belong to a meta-level. Also, there may be two groups of vehicles (`oil`/`electric` powered) that may include agents of both roles. Besides, there may be a relation of visibility between two cars.

$$roles = \{\mathtt{car}\} \tag{5.7}$$

$$groups = \{\mathtt{oil}, \mathtt{electric}\} \tag{5.8}$$

$$rels = \{Visibility(ag_i, ag_j) : \mathtt{car} \in \mathtt{R}_{ag_i} \wedge \mathtt{car} \in \mathtt{R}_{ag_j}\} \tag{5.9}$$

5.3.2 The Social Conventions

The social conventions ($SocConv$) consist of a set of interaction protocols ($Prots$), a set of norms ($Norms$) and some detection policy ($DetecPol$) —see eq. 5.10. In short, protocols define legitimate sequences of actions, norms limit agent's actions and/or determine their consequences, and the detection policy determines how to detect convention violations —see §2.4.2.

$$SocConv = Prots \times Norms \times DetecPol \tag{5.10}$$

In particular, a protocol ($Prot$) can be specified as a finite state machine model like in eq. 5.11-5.14. Hence, it has an input alphabet ($InputAlpha$) to define the events that can change its state among the set of possible states ($States$). Such events may be agent actions or illocutions, as well as environment events. Besides, state changes are described by a transition relation among states (T). Regarding the set of possible states, it includes an initial state ($IniState$), a set of final states ($EndStates$) and a current state determined at run-time —this execution state will also contain additional information such as which agents are participating.

$$Prots = \{p_1, \dots, p_{\#p} : p_i \in Prot\} \tag{5.11}$$

$$Prot = InputAlpha \times States \times IniState \times EndStates \times T \tag{5.12}$$

$$IniState \in States; \quad EndStates \subset States; \tag{5.13}$$

$$T = [States \times InputAlpha \rightarrow States] \tag{5.14}$$

In addition, an element of the set of norms ($Norms$) is a norm[6] ($Norm$) with a condition ($Cond$) expressed in deontic logic and optionally the consequences (Conseq) for violators —see eq. 5.15-5.16. The subset of deontic

[6] Notice that in our first approach (SAEI) we used the nomenclature of the Electronic Institution's related work [García-Camino et al., 2006], which named *rule* –see §3.2.3– to what we are referring as *norm* from now on. In a generic context, we prefer to use the term *norm* to refer to such concept, since it is currently widespread [Boella et al., 2010].

logic [Hilpinen et al., 1971] that we use lets express obligations (`obl`), permissions (`per`) and prohibitions (`prh`) over a certain predicate. Also, the consequence of a norm is a predicate in deontic logic that lets arise new obligations, permissions or prohibitions. From a more global perspective, the consequence may be used to enact an certain enforcement policy in form of punishments, rewards and/or impositions —see §2.4.2 for examples.

$$Norms = \{n_1 \dots n_{\#n} : n_i \in Norm\} \tag{5.15}$$

$$Norm = Cond \times Conseq \tag{5.16}$$

Finally, the detection policy (*DetecPol*) determines how to detect these violations and how to act (*ConseqPol*) in such cases —see eq. 5.17. In particular, the detection policies may include no detection (`none`), full detection by checking all actions (`full`) or partial detection (`partial`).

$$DetecPol = \{\texttt{none}, \texttt{full}, \texttt{partial}\} \tag{5.17}$$

Following the traffic example, the equations 5.18-5.21 detail the social conventions in this scenario. There may be a protocol that describes the valid sequence of actions required to turn in a crossroads (*turning*) —such protocol may concern the use of blinkers. Regarding norms, there may be one (*speedLimit*) limiting the speed of a car —we assume the *speed* is an agent observable property. Finally, the detection policy may be a partial detection of violations by police agents.

$$prots = \{turning\} \tag{5.18}$$

$$norms = \{speedLimit\} \tag{5.19}$$

$$speedLimit = \texttt{prh}\ (\texttt{car} \in \mathtt{R}_{ag_i} \wedge speed_{ag_i} > max_{speed}) \tag{5.20}$$

$$detecPol = \texttt{partial} \tag{5.21}$$

5.3.3 The Organisational Goals

The organisational goals (*Goals*) specification differs depending on the OCMAS approach —see §2.4.2. However, as this thesis focuses on regulation-oriented problems, our organisation model corresponds to a regulation definition approach —see §1.2.1. In particular, the organisational goals model is based on the AEI's approach —see §3.4.1. As defined by equations 5.22-5.24 they consist of a set of relevant observable properties (*PropGoals*) and an objective function (Ω) that measures an overall goal satisfaction from current observations. This goal satisfaction is measured as a real value among 0 (non-achieved goals) and 1 (achieved goals).

$$Goals = PropGoals \times \Omega \tag{5.22}$$

$$PropGoals \subseteq \left\{AgsP^S \cup EnvP^S\right\} \tag{5.23}$$

$$\Omega = \left[(\mathbb{R})^{|PropGoals|} \to \mathbb{R} \in [0..1] \right] \tag{5.24}$$

As an illustration in the traffic example, the goals may be: to have a fluid traffic flow and no collisions. In order to formalise these goals, we first define the observable properties and next we specify the goals.

As for the observable properties, it is necessary to observe if a car is waiting ($Wait$) and the number of collisions ($Collis$), in order to evaluate goal fulfilment. The former is a property that belongs to the observable properties of an agent (AgP). Whereas the latter is an observable property that belongs to the environment ($EnvP$). Accordingly, these properties are defined in equations 5.25-5.31. Moreover, these definitions include the speed of a car ($Speed$) used in §5.3.2 and the traffic density ($Density$) required in §5.5.2.3.

$$AgsP^S = \{agP_i \in AgP\}\,,\ \left|AgsP^S\right| = |Ags| \tag{5.25}$$

$$AgP = Wait \times Speed \tag{5.26}$$

$$Wait = \{\texttt{true}, \texttt{false}\} \tag{5.27}$$

$$Speed = s \in \mathbb{R} : s \geq 0 \tag{5.28}$$

$$EnvP^S = Collis \times Density \tag{5.29}$$

$$Collis = \mathbb{N} \tag{5.30}$$

$$Density = d \in \mathbb{R} : d \geq 0 \tag{5.31}$$

Once defined the observable properties, the equations 5.32-5.38 express the goals specification. Basically, there are two relevant observable properties ($Wait$, $Collis$). Their desired values are false for the former and zero for the latter. Accordingly, there is an objective function (ω) which computes the goal satisfaction as a weighted addition of these two criteria –defined as separate functions (γ_{Wait},γ_{Collis}) for simplicity– where the last one is the more relevant —see eq. 5.38. On the one hand, there is a function (γ_{Wait}) which computes the ratio of non-waiting cars ($wait_i$) over the number of cars. And, on the other hand, there is a function (γ_{Collis}) which computes the ratio of the number of collisions ($collis$) over the number of cars.

$$goals = (propGoals, \omega) \tag{5.32}$$

$$propGoals = \{waits, collis\} \tag{5.33}$$

$$waits = \bigcup_{i=1}^{|ags|} wait_i \in Wait \tag{5.34}$$

$$\omega(waits, collis) = k \cdot \gamma_{Wait}(waits) + (1-k) \cdot \gamma_{Collis}(collis) \tag{5.35}$$

$$\begin{aligned} \gamma_{Wait}\left(\{wait_i \in Wait\}\right) = &\ {}^{|\{wait_i : wait_i = \texttt{false}\}|}/_{|cars|} \\ cars = &\ \{ag_i : ag_i \in ags \wedge \texttt{car} \in \mathtt{R}_{ag_i}\} \end{aligned} \tag{5.36}$$

$$\gamma_{Collis}\left(collis\right) = {}^{(|cars| - collis)}/_{|cars|} \tag{5.37}$$

$$k = 0.2 \tag{5.38}$$

5.4 Features of agents in charge of adaptation

Our initial AOCMAS approach presented in chapter 3, the SAEI, is a centralised approach based on Electronic Institutions. Hence, in order to provide an enhanced approach, previous section generalises its organisational model. Now, this section identifies the features present in agents that can perform the organisational adaptation. As a result, next section 5.5 proposes the enhanced approach (2-LAMA).

Given the AOCMAS state of the art (see §2.5.3), we identified the following features in agents that are in charge of adaptation:

- Such agents are usually able *to gather certain information* that may not be available to all agents. For instance, in the traffic scenario, a police agent can access some traffic flow statistics (i.e. *Collis*, *Density*) that are not available to car agents.

- Such agents are able *to reason at a level of abstraction higher* than the required by domain activities. For example, a police agent requires a more general scope reasoning in order to establish traffic lights' intervals than the basic driving task performed by car agents.

- Such agents *consider system goals* beyond their individual goals, taking into account the social-welfare. In other words, such agents consider the organisational goals (*Goals*). For instance, in our traffic scenario, a police agent may take into consideration global traffic flow.

- Such agents are *empowered to update organisational structures* whereas other agents can only examine them (e.g. *Norms*). For example, a police agent can update traffic lights' intervals, whereas a car agent can only observe them.

Given all these agent features, in next section we propose to count on a meta-level set of agents that presents these features. Such agents are in charge of adapting –and thus, empowered to adapt– the organisation of the agents that participate in the domain activity. Accordingly, these meta-level agents are meant to reason at a higher level of abstraction than domain agents. Following a division of labour paradigm, instead of increasing the complexity of domain agents (ag_i), assistants are the ones in charge of taking into account organisational goals. Moreover, assistants act as trusted third parties, since they are not involved in domain activities and they are provided by the infrastructure.

5.5 Two-Level Assisted MAS Architecture (2-LAMA)

This section describes our AOCMAS enhanced approach (2-LAMA). It is an improvement on our first approach (SAEI) which had a centralised mechanism and

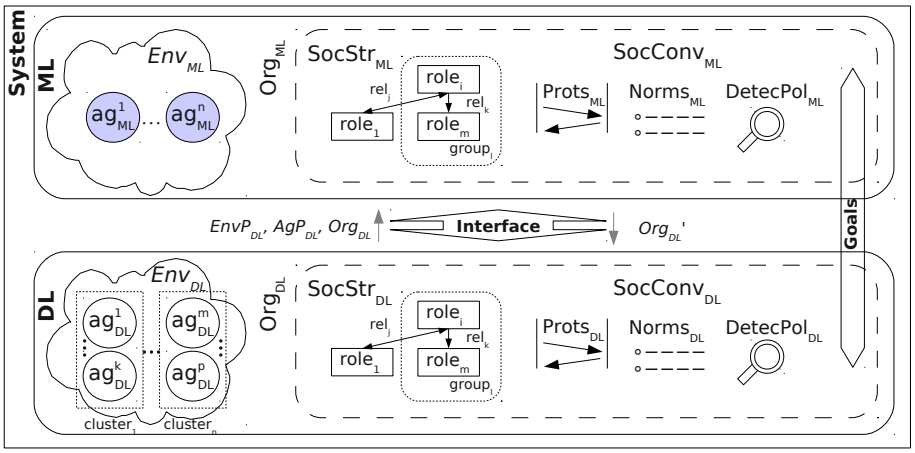


Figure 5.3: Two Level Assisted MAS Architecture (2-LAMA).

was based on an electronic institution. In contrast, 2-LAMA has a distributed adaptation mechanism and is based on a general organisation model.

First, this section defines 2-LAMA abstract architecture. Next, it formalises its organisational adaptation capabilities and its distributed implementation. Finally, it examines the costs and frequency of such adaptation.

5.5.1 Abstract Architecture

In order to improve SAEI (see ch. 3), our enhanced approach is based on the general organisation model proposed in §5.3. As a result, we call it an *abstract architecture* since it can be applied to different specific organisation models. Furthermore, its assistance services are distributed among a meta-level set of agents as depicted in figure 5.3. Accordingly, we call our enhanced approach Two Level Assisted MAS Architecture (2-LAMA), which is defined as follows:

Definition 6 *We define a Two Level Assisted MAS Architecture (2-LAMA) as*

$$2LAMA = ML \times DL \times \Gamma \tag{5.39}$$

where:

- *ML stands for a Meta-Level, which is the part of the system that provides the assistance services.*
- *DL stands for a Domain-Level, which is the part of the system related to domain activity.*
- Γ *stands for the set of functions that describe the assistance services.*

Notice that, it is possible to nest subsequent meta-levels to provide assistance to previous level's organisation.

Each level has its own set of agents (Ag_{ML}, Ag_{DL}), organisation (Org_{ML}, Org_{DL}) and environment (Env_{ML}, Env_{DL}) as shown in equations 5.40-5.41. Notice that ML an DL subscripts are introduced to distinguish system main components among those two levels. Nevertheless, for the sake of notation simplicity, in subsequent sections these subscripts are omitted where there are no ambiguities when referring to them.

$$ML = Ag_{ML} \times Org_{ML} \times Env_{ML} \tag{5.40}$$

$$DL = Ag_{DL} \times Org_{DL} \times Env_{DL} \tag{5.41}$$

On the other hand, the assistance functions (Γ) are described in subsection §5.5.2.

5.5.1.1 Meta-Level

Since agents at meta-level provide assistance services, we henceforth refer to them as *assistants*. These agents present the features previously identified in §5.4. Hence, they are able to gather information from DL in order to reason about it by taking into account DL's organisational goals. Even more, they are empowered to update the organisational structures of DL.

Accordingly, the meta-level's organisational goals ($Goals_{ML}$) come down to help the domain level to achieve its own goals ($Goals_{DL}$). Hence, they are equivalents as expressed in equation 5.42 and depicted in in figure 5.3.

$$Goals_{ML} \equiv Goals_{DL} \tag{5.42}$$

Also, ML's environment contains DL's observable properties about its agents and environment (AgP^S_{DL}, $EnvP^S_{DL}$), as well as, its explicit organisation specification (Org_{DL}) as shown in equation 5.43.

$$Env_{ML} = EnvP_{ML} \times AgsP^S_{DL} \times Org_{DL} \times EnvP^S_{DL} \tag{5.43}$$

Nevertheless, locality –a fundamental feature of any MAS– is also applied to this architecture. In this manner, assistants are just "in charge of" assisting a subset (a *cluster* hereafter) of domain-level agents. This leads to a partial information assumption, where assistants only perceive the observable properties of both its assisted agents and the local environment where they are situated.

5.5.1.2 Domain-Level

Domain-Level agents are regular MAS participants, which perform the domain activity. In general, as equation 5.44 shows, agents can be characterised by a set of properties. All agents have the same kind of properties (i.e. an agent has the $\#ag\_prop$ properties defined by the AgP model), but with different values. Furthermore, as stated above, an assistant is able to observe the properties of all the domain agents in its cluster ($AgsP^C$, eq. 5.45). Such clusters are defined depending on domain-specific criteria. Similarly, equation 5.46 expresses the set

of system-wide observed properties ($AgsP^S$) as the union of properties observed along clusters.

Regarding environment properties, we use an analogous notation to denote the properties of the environment region where a cluster of agents are located ($EProp_y$ in $EnvP^C$), and how they are aggregated at system level ($EnvP^S$) —see eq. 5.47 and 5.48. Notice that there may be also some environmental observable properties that are not associated to any cluster in particular ($EnvP^{SnC}$) —e.g. time. Hence, these properties ($EProp_y^{SnC} \in EnvP^{SnC}$) are also present at system-level as shown in equations 5.49 and 5.48. Overall, in some expressions we use a single identifier ($ObsProp$) to refer to all observable properties, which is defined in equation 5.50.

$$AgP = AProp_1 \times \ldots \times AProp_{\#ag\_prop} \tag{5.44}$$

$$AgsP^C = \bigcup_{j=1}^{m_i} AgP_j \tag{5.45}$$

$$AgsP^S = \bigcup_{i=1}^{n} AgsP_i^C \tag{5.46}$$

$$EnvP^C = EProp_1 \times \ldots \times EProp_{\#env\_prop} \tag{5.47}$$

$$EnvP^S = \bigcup_{i=1}^{n} EnvP_i^C \cup EnvP^{SnC} \tag{5.48}$$

$$EnvP^{SnC} = EProp_1^{SnC} \times \ldots \times EProp_{\#EnvP^{SnC}}^{SnC} \tag{5.49}$$

$$ObsProp_{DL} = AgsP_{DL} \cup EnvP_{DL} \tag{5.50}$$

5.5.1.3 Discussion

So far we have presented an abstract architecture containing a separated assistance layer (the meta-level) with a distributed design. Separation of concerns and distribution are two design decisions that follow the MAS paradigm —see Agent Oriented Software Engineering (AOSE) in [Wooldridgey and Ciancarini, 2001]. Therefore, they also benefit from the same advantages of robustness and the absence of global-information requirements. Separation of concerns allows assistants to reason at a higher level of abstraction than domain-level agents since they can be completely devoted to the processes of summarising and sharing local data. This is also beneficial for DL agents, since they do not need to increase their reasoning complexity.

Additionally, having assistants separated, allows to grant certain information privileges such as having access to environmental properties (since it is not always the case that, for example, cars have access to the average traffic flow density). Similarly, their decision making involves system goals properties that may not be available to domain-level agents. Furthermore, they can individually

specialize to provide specific services or to reorganise their society so to best fit the heterogeneity and dynamism of the needs of assisted agents.

Finally, other agents should regard them as trusted third-parties when accepting their assistance or revealing information. Regarding distribution, it requires agent communication, our proposal minimizes its costs by keeping most of it local to clusters.

In conclusion, the proposed architecture assumes assistants have the features described in §5.4. In order to fulfil these requirements, we have chosen an implementation for this abstract architecture that defines assistants as *staff agents* which belong to the organisation.

5.5.1.4 Example

Considering our traffic example, we provide a *particular specification* ($2lama_{traff}$) of the 2-LAMA model ($2LAMA$) as shown in equation 5.51. According to our 2-LAMA model, such specification has its own meta-level ($ml_{traff} \in ML$), domain-level ($dl_{traff} \in DL$) and assistance functions ($\gamma_{traff} \in \Gamma$) as expressed in equation 5.52. For the sake of simplicity, in the following equations we omit the 'traff' subscript since all of them refer to the traffic scenario. Even more, we omit to explicitly indicate that each symbol in lower-case letters belongs to the upper-case symbol defined by the model (e.g. $dl \in DL$).

$$2lama_{traff} \in 2LAMA \tag{5.51}$$

$$2lama_{traff} = (ml_{traff}, dl_{traff}, \gamma_{traff}) \tag{5.52}$$

$$dl = (ags_{DL}, org_{DL}, env_{DL}) \tag{5.53}$$

$$ml = (ags_{ML}, org_{ML}, env_{ML}) \tag{5.54}$$

The *domain-level* in the traffic scenario (dl) has its own agents (ags_{DL}) interacting within and environment (env_{DL}) according to a certain organisation (org_{DL}) as appear in eq. 5.53. In particular, its organisation is specified along the examples of §5.3 (eq. 5.7-5.38). Moreover, that section also specifies the observable properties of their agents ($AgsP^S$ in eq. 5.25) and environment ($EnvP^S$ in eq. 5.29). Whereas the particular agents (ags_{DL}) and environment (env_{DL}) will be actually defined at run-time (i.e. at the Execution State stage, see §5.2).

Analogously, the *meta-level* (ml) has also its own agents (ags_{ML}), environment (env_{ML}) and organisation (org_{ML}) as appear in eq. 5.54. In this scenario, the police agents are the ones that constitute the meta-level since they are not participating in the domain activity (i.e. driving from one origin to one destination), but assisting the coordination of such activity. Hence, there is a single role in the meta-level organisation (org_{ML}) as defined in eq. 5.55. As a basic illustration, there are no groups among policemen (eq. 5.56) but there is a relation among them (eq. 5.57). It represents a radio link ($radioLink$) among two policemen which means they can exchange information about the observable properties they perceive —i.e. it constitutes their net of relationships. In fact, they use a certain communication protocol ($walkieTalkie$) when using this radio

link. Hence, their social conventions include this protocol (eq. 5.58). However, in this example it does not include any norm (eq. 5.59) nor detection policy (eq. 5.60) since they are system trusted agents (i.e. staff agents). Finally, as stated in 2-LAMA model (eq. 5.42), their organisational goals are shared with domain-level as shown in eq. 5.61.

$$roles_{ML} = \{\texttt{police}\} \tag{5.55}$$

$$groups_{ML} = \emptyset \tag{5.56}$$

$$rels_{ML} = \left\{radioLink(ag_i, ag_j) : \texttt{police} \in \texttt{R}_{ag_i} \wedge \texttt{police} \in \texttt{R}_{ag_j}\right\} \tag{5.57}$$

$$prots_{ML} = \{walkieTalkie\} \tag{5.58}$$

$$norms_{ML} = \emptyset \tag{5.59}$$

$$detecPol_{ML} = \texttt{none} \tag{5.60}$$

$$goals_{ML} = goals_{DL} \tag{5.61}$$

The remaining assistance functions specification (γ_{traff}) is illustrated at the end of next subsection —specifically in §5.5.2.3.

5.5.2 Assistance Functions

In addition to specifying the Meta-Level and Domain-Level, the 2-LAMA model includes the definition of the assistance functions (Γ, see eq. 5.39). Such functions were briefly introduced when formalising assistance services in chapter 4. Accordingly, Γ can be expressed as equation 5.62. In particular, our main target is to provide the Organisational Adaptation service (F^{adapt}, eq. 4.1). Accordingly, this section provides further details about this service and its formalisation as a set of functions.

5.5.2.1 Formalisation

For the sake of notation simplicity, henceforth we refer to the corresponding set of functions as an upper-case alpha 'α' with an 'O' superscript[7] as shown in equation 5.63. Hence, the *adaptation of an organisation* can be expressed as a function (α^O) that provides an updated organisation (Org) depending on both the system's observable properties ($EnvP^S$ and AgP^S) and current organisation —see eq. 5.63 and previous eq. 4.1.

$$\Gamma = \Gamma^{info} \times \Gamma^{justif} \times \Gamma^{advice} \times \Gamma^{estim} \times \Gamma^{adapt} \tag{5.62}$$

$$\alpha^O = \Gamma^{adapt} \tag{5.63}$$

$$\alpha^O = \left[AgsP^S \times EnvP^S \times Org \rightarrow Org\right] \tag{5.64}$$

[7] Notice that 'α' stands for 'adaptation', whereas the superscript denotes what is being updated (e.g. the organisation is represented by an 'O').

Depending on the organisation design, the adaptation of its components may be totally dependent, partially related or completely independent. The more dependent they are, the more information is required when making adaptation decisions. The proposed driving force behind adaptation is goal accomplishment, and so *Goals* are considered in the adaptation functions of all organisational components. Therefore, we are taking an assumption of partial-relation. If all components were dependent, then the whole organisation would be required to be considered when adapting. Obviously, such a case would also imply an increase of the complexity of the adaptation function.

Accordingly, we define the adaptation of the Social Structure (α^{SS}, see eq. 5.65) as a function that provides an updated social structure depending on system's status –i.e. its observable properties–, the defined goals and current social structure. Analogously, we define the adaptation of the Social Conventions (α^{SC}, see eq. 5.69) and a function to adapt each one of its components. Specifically, one function for the adaptation of interaction Protocols (α^{P}), another one for the adaptation of Norms (α^{N}) and a last one for the adaptation of the Detection Policy (α^{D}) as defined in equations 5.70-5.72. In the same manner, we define the adaptation of Goals (α^{G}) as the function expressed in equation 5.73.

$$\alpha^{SS} = \left[AgsP^S \times EnvP^S \times Goals \times SocStr \rightarrow SocStr\right] \tag{5.65}$$

$$\alpha^{R} = \left[AgsP^S \times EnvP^S \times Goals \times Roles \rightarrow Roles\right] \tag{5.66}$$

$$\alpha^{Gr} = \left[AgsP^S \times EnvP^S \times Goals \times Groups \rightarrow Groups\right] \tag{5.67}$$

$$\alpha^{Rels} = \left[AgsP^S \times EnvP^S \times Goals \times Rels \rightarrow Rels\right] \tag{5.68}$$

$$\alpha^{SC} = \left[AgsP^S \times EnvP^S \times Goals \times SocConv \rightarrow SocConv\right] \tag{5.69}$$

$$\alpha^{P} = \left[AgsP^S \times EnvP^S \times Goals \times Prots \rightarrow Prots\right] \tag{5.70}$$

$$\alpha^{N} = \left[AgsP^S \times EnvP^S \times Goals \times Norms \rightarrow Norms\right] \tag{5.71}$$

$$\alpha^{D} = \left[AgsP^S \times EnvP^S \times Goals \times DetecPol \rightarrow DetecPol\right] \tag{5.72}$$

$$\alpha^{G} = \left[AgsP^S \times EnvP^S \times Goals \rightarrow Goals\right] \tag{5.73}$$

Overall, when having a specific organisation $org \in Org$, its adaptation is defined as follows:

Definition 7 *Given an organisation* $org = (socstr, socconv, goals) \in Org$ *where* $socconv = (prots, norms, enfPol)$*, current values of environment properties* $envp^S$*, and current values of agent properties* agp^S*, we define the organisation adaptation of org as:*

$$\begin{aligned}
\alpha^O\left(envp^S, agp^S, org\right) &= org', where: \\
org' &= ((roles, groups, rels), (prot', norms', enfPol'), goals') \\
roles' &= \alpha^R\left(envp^S, agp^S, goals, roles\right) \\
groups' &= \alpha^{Gr}\left(envp^S, agp^S, goals, groups\right) \\
rels' &= \alpha^{Rels}\left(envp^S, agp^S, goals, rels\right) \\
prots' &= \alpha^P\left(envp^S, agp^S, goals, prots\right) \\
norms' &= \alpha^N\left(envp^S, agp^S, goals, norms\right) \\
detecPol' &= \alpha^D\left(envp^S, agp^S, goals, detecPol\right) \\
goals' &= \alpha^G\left(envp^S, agp^S, goals\right)
\end{aligned} \tag{5.74}$$

Assuming that: $\alpha^O \in \alpha^O$, $\alpha^{SS} \in \alpha^{SS}$, $\alpha^{SC} \in \alpha^{SC}$, $\alpha^P \in \alpha^P$, $\alpha^N \in \alpha^N$, $\alpha^D \in \alpha^D$, $\alpha^G \in \alpha^G$.

5.5.2.2 Discussion

Basically, these adaptation functions evaluate the current system's status in order to modify specific organisational components. As mentioned above, these changes are driven by system goals, so that changes are introduced with the aim of inducing a higher accomplishment of current goals. Our proposal is that assistants in the meta-level apply these adaptation functions when agents at the domain-level fail to obtain the desired performance. In this sense, it can be interpreted as a top-down adaptation approach. Nevertheless, the meta-level could also be sensitive to changes in the agent population that may require fundamental changes —such as goal adaptation. This case is closer to a bottom-up approach, and thus, a hybrid adaptation approach may be more flexible.

Notice that the goals adaptation function (α^G) requires special attention since its outcomes affect the rest of adaptation functions (see subsequent §5.5.4 about adaptation frequency for a further discussion on that). More importantly, it may change fundamental design goals, and so, some basic system properties should be guaranteed by means of additional mechanisms such as specific goal updating policies. Although this discussion goes beyond the scope of this thesis, we envision that measures related to the number of convention violations –which may be related to agents' degree of satisfaction– may motivate reconsidering some goals.

5.5.2.3 Example

In general, the assistance functions are already illustrated in chapter 4. Nevertheless, this section provides further examples about the adaptation function (α^O or Γ^{adapt}, see eq. 5.64) in the traffic scenario. In particular, we provide some brief descriptions about the adaptation functions of each organisational component (eq. 5.65-5.73). Besides, next chapters 6 and 7, provide more examples in the P2P scenario, including full details about the norm adaptation function.

As an illustration, the traffic example counts on a simple social structure adaptation function ($\alpha^{SS} \in \alpha^{SS}$) defined in eq. 5.75. Basically, in case there

are more collisions than a certain threshold ($\texttt{max}_{\texttt{collis}}$), it adds and ambulance role (`ambulance`) to help restoring the traffic flow by picking the injured drivers (`pickInjured`). Otherwise, it removes such new role and relationship.

$$\begin{aligned}
\alpha^{SS} \quad & \left(agsp^S, envp^S, goals, socStr\right) = socStr' \\
& socStr' = \begin{cases} (roles', groups, rels') & if\ (collis > \texttt{max}_{\texttt{collis}})\,,\ collis \in envp^S \\ (roles'', groups, rels'') & otherwise \end{cases} \\
& roles' = roles \cup \{\texttt{ambulance}\} \\
& rels' = rels \cup \{pickInjured\,(ag_i, ag_j) : \texttt{ambulance} \in \texttt{R}_{ag_i} \wedge \texttt{car} \in \texttt{R}_{ag_j}\} \\
& roles'' = \{\texttt{ambulance}\} \backslash roles = roles \cap \{\texttt{ambulance}\}^C \\
& rels'' = \{pickInjured\,(ag_i, ag_j) : \texttt{ambulance} \in \texttt{R}_{ag_i} \wedge \texttt{car} \in \texttt{R}_{ag_j}\} \backslash rels
\end{aligned} \tag{5.75}$$

On the other hand, there is also a simple social conventions adaptation function ($\alpha^{SC} \in \alpha^{SC}$) depicted in eq. 5.76. Essentially, it only adapts the protocols and norms by using their adaptation functions defined in equations 5.77-5.78. In particular, the protocol adaptation function ($\alpha^P \in \alpha^P$) sets a safer turning procedure (e.g. it includes safety distances to prevent collisions) in case there are more collisions than the mentioned threshold. Besides, the norm adaptation function ($\alpha^N \in \alpha^N$) is able to decrease the speed limit to avoid collisions when there is a large density (larger than a $\texttt{max}_{\texttt{density}}$ threshold).

$$\begin{aligned}
\alpha^{SC} \quad & \left(agsp^S, envp^S, goals, socConv\right) = socConv' \\
& socConv' = \begin{array}{l} \left(\alpha^P\left(agsp^S, envp^S, goals, prots\right),\right. \\ \alpha^N\left(agsp^S, envp^S, goals, norms\right), \\ \left. detecPol\right) \end{array}
\end{aligned} \tag{5.76}$$

$$\begin{aligned}
\alpha^{P} \quad & \left(agsp^S, envp^S, goals, prots\right) = prots' \\
& prots' = \begin{cases} (\{turning\} \backslash prots) \cup \{turningSafer\} & if\ (collis > \texttt{max}_{\texttt{collis}}) \\ (\{turningSafer\} \backslash prots) \cup \{turning\} & otherwise \end{cases}
\end{aligned} \tag{5.77}$$

$$\begin{aligned}
\alpha^{N} \quad & \left(agsp^S, envp^S, goals, norms\right) = \left\{speedLimit : max_{speed} = max'_{speed}\right\} \\
& max'_{speed} = \begin{cases} \texttt{lowSpeed} & if\ \left(density > \texttt{max}_{\texttt{density}}\right),\ density \in envp^S \\ \texttt{highSpeed} & otherwise \end{cases}
\end{aligned} \tag{5.78}$$

Finally, a goal adaptation function ($\alpha^G \in \alpha^G$) updates organisational goals as expressed in eq. 5.79. Specifically, it gives more weight to traffic flow than to collisions (see eq. 5.35) when the average speed of cars (*speed*) is lower than a certain threshold ($\texttt{min}_{\texttt{speed}}$).

$$\begin{aligned}
\alpha^{G} \quad & \left(agsp^S, envp^S, goals\right) = goals : k = k' \\
& k' = \begin{cases} 0.8 & if\ \left(speed < \texttt{min}_{\texttt{speed}}\right),\ speed \in agsp^S \\ 0.2 & otherwise \end{cases}
\end{aligned} \tag{5.79}$$

5.5.3 Distributed adaptation

In our 2-LAMA approach, the adaptation function (α^O) defined above is performed by the meta-level in order to adapt the domain-level's organisation. The distributed nature of meta-level raises some issues to take into account. Each assistant perceives partial information about system status and computes a summary that shares subsequently with other assistants. Besides, organisation adaptation is distributed between assistants. In this manner, each assistant computes the desired adaptations for each organisational component, and later on, their adaptation proposals have to be combined to end up with new organisational configurations. The rest of this subsection is devoted to exploring each of these issues.

5.5.3.1 Information

In 2-LAMA, each assistant perceives information about the cluster of agents it assists and about the corresponding environment —see §5.5.1.1. We refer to this information as assistant's *local information* ($agsp_i^C \in AgsP^C$, $envp_i^C \in EnvP^C$, i being the assistant's index). Afterwards, it shares a summary of this information with other assistants. Thus, we define as *remote assistant information* all pieces of summary information received from other assistants ($sump_1, \ldots, sump_{i-1}, sump_{i+1}, \ldots, sump_n$). This way, each assistant has an abstraction of overall information when taking its decisions. Finally, we define the *knowledge* of an assistant ($knowp_i$) as the aggregation of its local and remote pieces of information including its perception of those environment observable properties that are not associated to any cluster in particular ($EnvP^{SnC}$).

This modelling requires us to define two processes: how local information is summarised and how an assistant aggregates its local and remote information. First, we define a summary function (Σ, eq. 5.80) which constructs a summary ($sump_i \in SumP$[8], eq. 5.81) out of an assistant's local information. Thus, statistical functions, such as mean or average, are good candidates for summary functions. Second, we define the aggregation function (Λ, eq. 5.82) as the process that combines an assistant's local information with pieces of remote information (i.e. a set of summaries) and the information not associated to any cluster in particular ($envp^{SnC}$) to obtain an assistant's knowledge ($knowp_i$, see eq. 5.83). This knowledge is of type $KnowP$ and will be used in subsequent adaptation functions.

$$\Sigma = \left[AgsP^C \times EnvP^C \rightarrow SumP\right] \tag{5.80}$$

$$sump_i = \sigma(agsp_i^C, envp_i^C), \quad sump_i \in SumP, \quad \sigma \in \Sigma \tag{5.81}$$

$$\Lambda = \left[AgsP^C \times EnvP^C \times EnvP^{SnC} \times (SumP)^{n-1} \rightarrow KnowP\right] \tag{5.82}$$

$$\begin{gathered} knowp_i = \lambda(agsp_i^C, envp_i^C, envp^{SnC}, \{sump_x : x = 1..n \wedge x \neq i\}) \\ knowp_i \in KnowP, \quad \lambda \in \Lambda \end{gathered} \tag{5.83}$$

[8] Notice that $SumP$, the type of this information summary, is not qualified by the cluster superscript "C" because there is no need to differentiate it from $SumP^S$.

Finally, it is worth noticing that, although previous formulae assume assistants receive remote information from all other assistants (i.e. $\{sump_x | x \neq i\}$), it is not required to be the case. Actually, it depends on: whether or not the meta-level's social structure is fully connected, the reliability of communications, and the assistants' capacity to gather local information. Obviously, the lack of information may affect assistant's (and thus ML's) performance.

5.5.3.2 Decision making

Distribution at meta-level concerns both information and decision making. As mentioned above, assistants initially make their individual decisions based on the available information and the system's goals. Afterwards, they reach an agreement over the actual domain-level organisational changes. The equations below illustrate this process for the norm adaptation —besides, their cost and frequency is analysed in §5.5.4. First, as shown in equation 5.84, we define the decision making of a single assistant i, as a partial adaptation function (α_i^N) with a similar domain and codomain than the meta-level adaptation function previously introduced (α^N, eq. 5.71). In particular, when an assistant i applies its adaptation function $\alpha_i^N \in \alpha_i^N$, it uses its knowledge, the system's goals and organisational norms to make its own decision about the definition of new norms. Afterwards, all assistants perform an *agreement process* by means of a function (β_{α^N}) which takes as many norm update proposals as the number of assistants and generates the actual norm update as described in equation 5.85.

$$\alpha_i^N = [KnowP \times Goals \times Norms \rightarrow Norms] \tag{5.84}$$

$$\beta_{\alpha^N} = [(Norms)^n \rightarrow Norms] \tag{5.85}$$

Previous equation 5.85 aggregates n different decisions because it assumes norms are global –at domain-level– and relevant to all assistants. [9]Taking this into consideration, equation 5.86 shows the overall meta-level norm adaptation function (α^N) as the agreement of partial norm adaptation functions α_i^N individually computed by assistants.

$$\begin{aligned} \alpha^N \quad & (envp, agp, goals, norms) = \\ & = \beta_{\alpha^N}(\alpha_1^N(knowp_1, goals, norms), \ldots, \alpha_n^N(knowp_n, goals, norms)) \\ & \alpha^N \in \alpha^N, \quad \alpha_i^N \in \alpha_i^N, \quad \beta_{\alpha^N} \in \beta_{\alpha^N} \end{aligned} \tag{5.86}$$

Regarding the other organisational components' related functions –i.e. partial adaptations (α_i^{SS}, α_i^P, α_i^{EP}, α_i^G), agreement processes ($\beta_{\alpha^{SS}}$, β_{α^P}, $\beta_{\alpha^{EP}}$, β_{α^G}) and adaptation functions (α^{SS}, α^P, α^{EP}, α^G)– their domain/codomain and definitions are analogous to equations 5.84-5.86. They use the knowledge derived from exchanged summaries and the organisational goals to

[9]Nevertheless, it could be the case that certain norms apply only to certain contexts, and thus, just affected assistants should agree upon their update. Taking that to the limit, it may be the case that a single involved assistant does not need to agree with anyone else.

compute the corresponding updated organisational component —notice though, that the domain of goal related functions is simpler, since it does not consider any other organisational components.

5.5.3.3 Example

Following the traffic example, an assistant in this scenario is in charge of a region of a road-network. Hence, it exchanges information about these regions ($sumP_i \in SumP$) with the other assistants in order to build its knowledge about the overall system status. Then, it can use this knowledge ($knowP_i \in KnowP$) to suggest organisational changes depending on its partial adaptation functions ($\alpha_i^z \in \alpha_i^z$, where z stands for any of the organisational adapted components). Finally, all assistants achieve an agreement ($\beta_{\alpha^z} \in \beta_{\alpha^z}$) on how the organisation is actually updated.

On the one hand, the local information of an assistant consists of the $AgsP$ and $EnvP$ defined by equations 5.26 and 5.29 respectively —in this example we do not consider general environmental properties ($EnvP^{SnC}$, eq. 5.49). On the other hand, the information required by the illustrative adaptation functions (eq.5.75-5.79) is related to the traffic density, the number of collisions and the speed of cars. Consequently, the summary information they exchange ($SumP$) includes only this sort of information (their identifiers have an 'S' prefix to denote 'summary') as defined in equation 5.87. Moreover, given that in this example there is no $EnvP^{SnC}$, the knowledge they use has also the same sort of information (they have the identifiers with a 'K' prefix) as shown in equation 5.88. In both cases, the values are positive real numbers (see eq. 5.89-5.90) since they correspond to the averages of such properties.

$$SumP = SDensity \times SCollis \times SSpeed \times SWait \tag{5.87}$$

$$KnowP = KDensity \times KCollis \times KSpeed \times KWait \tag{5.88}$$

$$PosR = \{n \in \mathbb{R} : n \geq 0\} \tag{5.89}$$

$$\begin{gathered} SDensity \in PosR,\ SCollis \in PosR,\ SSpeed \in PosR,\ SWait \in PosR \\ KDensity \in PosR,\ KCollis \in PosR,\ KSpeed \in PosR,\ KWait \in PosR \end{gathered} \tag{5.90}$$

Accordingly, an assistant computes the summary of its local information by using the summary function ($\sigma \in \Sigma$). This function provides the traffic flow density and the number of collisions in this region as well as computes the average speed of the cars traversing it and the number of waiting cars as expressed in equation 5.91.

$$\begin{aligned} &\sigma(agp_i^C, envp_i^C) = sump_i = \\ &\quad \big(density \in envp_i^C, \\ &\quad\ collis \in envp_i^C, \\ &\quad\ avg\left(\{speed_{i,j} \in agp_i^C : j = 1..m_i\}\right), \\ &\quad\ \left|\{wait_{i,j} \in agp_i^C : j = 1..m_i : wait_{i,j} = \texttt{false}\}\right|\big) \end{aligned} \tag{5.91}$$

Next, an assistant uses the aggregation function ($\lambda \in \Lambda$) to build its knowledge from the received summaries. Specifically, in our example this function computes a weighted average of each summary of local information and all received remote information as defined in equation 5.92. Therefore, depending on the constant weights applied to local ($\mathtt{w_L}$) and remote ($\mathtt{w_{R,x}}$) information, an assistant can give more or less relative relevance to each one.

$$\begin{aligned}
&\lambda\left(agp_i^C, envp_i^C, envp^{SnC}, \{sump_x : x = 1..n \wedge x \neq i\}\right) = knowp_i = \\
&\quad = (kDensity_i, kCollis_i, kSpeed_i, kWait_i) \\
&\quad kDensity_i = \mathtt{w_L} \cdot sDensity_i + \textstyle\sum_{x=1}^{n} \{\mathtt{w_{R,x}} \cdot sDensity_x : x \neq i\}, \\
&\qquad\qquad sDensity_i \in envp_i^C,\ sDensity_x \in sump_x \\
&\quad kCollis_i = \mathtt{w_L} \cdot sCollis_i + \textstyle\sum_{x=1}^{n} \{\mathtt{w_{R,x}} \cdot sCollis_x : x \neq i\}, \\
&\qquad\qquad sCollis_i \in envp_i^C,\ sCollis_x \in sump_x \\
&\quad kSpeed_i = \mathtt{w_L} \cdot sSpeed_i + \textstyle\sum_{x=1}^{n} \{\mathtt{w_{R,x}} \cdot sSpeed_x : x \neq i\}, \\
&\qquad\qquad sSpeed_i \in agsp_i^C,\ sSpeed_x \in sump_x \\
&\quad kWait_i = \mathtt{w_L} \cdot sWait_i + \textstyle\sum_{x=1}^{n} \{\mathtt{w_{R,x}} \cdot sWait_x : x \neq i\}, \\
&\qquad\qquad sWait_i \in agsp_i^C,\ sWait_x \in sump_x
\end{aligned} \tag{5.92}$$

Once an assistant has the knowledge, it uses its partial adaptation functions ($\alpha_i^z \in \boldsymbol{\alpha}_i^z$) to decide how to update the organisation. In order to compute the general adaptation functions described in §5.5.2.3, the partial functions have an equivalent definition so that the general function is the result of applying the agreement function. For instance, the partial norm adaptation function (α_i^N) defined in eq. 5.93 is equivalent to the idea expressed in eq. 5.78, but obtains the density ($kDensity$) from knowledge ($knowp_i$). Hence, the general norm adaptation function (α^N) is actually the result of applying the agreement function ($\beta_{\boldsymbol{\alpha}^N}$) to the result of all the partial functions as shown in eq. 5.94.

$$\begin{aligned}
\alpha_i^N \quad & (knowp_i, goals, norms) = \left\{speedLimit : max_{speed} = max'_{speed}\right\} \\
& max'_{speed} = \begin{cases} \mathtt{lowSpeed} & if\ \left(kDensity > \mathtt{max_{density}}\right),\ kDensity \in knowp_i \\ \mathtt{highSpeed} & otherwise \end{cases}
\end{aligned} \tag{5.93}$$

$$\begin{aligned}
\alpha^N \quad & \left(agsp^S, envp^S, goals, norms\right) = \\
& = \beta_{\boldsymbol{\alpha}^N}(\alpha_1^N(knowp_1, goals, norms), \ldots, \alpha_n^N(knowp_n, goals, norms))
\end{aligned} \tag{5.94}$$

$$\beta_{\boldsymbol{\alpha}^N} = voting \tag{5.95}$$

In particular, in our example, the agreement follows a voting scheme (eq. 5.95). That is to say, each assistant sends its proposal change for each organisational component to the rest of assistants (its votes). As a consequence, an assistant receive all the proposals (i.e. the votes) from the rest of assistants. So, it can compute the most voted option for each component separately —notice, this process is replicated, since all assistants recount the same votes. Finally, the winner updates for each organisational components are applied.

5.5.4 Costs and frequency

The process of the organisational adaptation (α^O) involves some associated costs –in time and/or resources– that should be considered when defining the adaptation frequency. That is to say, the resulting frequency should keep the adaptation costs below the benefits it generates.

5.5.4.1 Adaptation Costs

In particular, as equation 5.96 shows, the organisational adaptation cost (c_{α^O}) comprises an information retrieval cost (c_{info^O}), a computation cost (c_{comp^O}), an adoption cost (c_{adopt^O}) and a transition cost (c_{trans^O}).

$$c_{\alpha^O} = c_{info^O} + c_{comp^O} + c_{adopt^O} + c_{trans^O} \tag{5.96}$$

The former, the *information retrieval cost* (c_{info^O}), is related to the cost of collecting the information required by the adaptation function. For example, collecting $AgsP$ may require some time and resources to exchange messages between assistants and participant agents. The second cost, the *computation cost* (c_{comp^O}), reflects the time and resources required to compute the adaptation function. That is to say, the time required to compute all α_i^N in parallel plus the cost of achieving an agreement (β_{α^N}) at meta-level. The third cost, the *adoption cost* (c_{adopt^O}), is related to the cost of transforming the previous organisation into the adapted one. As an illustration, when a norm is updated, messages must be sent to inform agents and they may need time to alter their activity in order to comply with the new norm. The last cost, the *transition cost* (c_{trans^O}), is the time and resources required for system's stabilisation. In this manner, the new organisation can be evaluated without interference from the previous one, since the effects of previous norms may persist longer in the environment than their actual activation period.

Furthermore, as the organisational adaptation function (α^O) is defined by the adaptation functions of all its components (see §5.5.2.3), its cost depend on a combination (f) of the costs of each adaptation function. For example, as shown in equation 5.97, the information retrieval cost derives from the costs of collecting information to perform: the social structure adaptation ($c_{info^{SS}}$), the adaptation of social conventions ($c_{info^{SC}}$) and the goal adaptation (c_{info^G}). Notice that in equation 5.97 there is not an addition but a function of those costs since there may be retrieved information that is useful to adapt more than one component. Thus, the cost of retrieving such shared information may be present in different component associated costs (e.g. $c_{info^{SS}}$, $c_{info^{SC}}$, c_{info^G}), but only needs to be added once to the final cost (e.g. c_{info^O}).

$$c_{info^O} = f(c_{info^{SS}}, c_{info^{SC}}, c_{info^G}) \tag{5.97}$$

5.5.4.2 Adaptation Frequency

The adaptation frequency ($freq$) of each of these organisational components can be chosen depending on its associated costs —and benefits. For instance, as equation 5.98 shows, the cost of retrieving information in order to perform the social structure adaptation ($c_{info^{SS}}$) is the sum of the cost of collecting information every time (x) the $SocStr$ is adapted. In this equation, the number of added terms ($\#\alpha^{SS}$) is the total number of performed $SocStr$ adaptations, which depends on the frequency[10] of adapting the social structure ($freq_{\alpha^{SS}}$) and the system's execution time (t). Thus, the higher the frequency is, the higher becomes the number of performed adaptations and their associated cost.

$$c_{info^{SS}} = \sum_{x=1}^{\#\alpha^{SS}} c_{info_x^{SS}} \ , \quad \#\alpha^{SS} = freq_{\alpha^{SS}} \cdot t \tag{5.98}$$

Although organisational components may adapt at different frequencies, it is important to ensure that goals are adapted with the lowest frequency —see eq. 5.99. In this way, the rest of the adaptation functions may have enough time to update their corresponding organisational components to current goals before they change. In this manner the period between goal adaptations should be long enough to allow the consequences of other component adaptations to emerge. Thus, the other adaptation functions may see the effects.

$$\left(freq_{\alpha^{G}} < freq_{\alpha^{SS}}\right) \wedge \left(freq_{\alpha^{G}} < freq_{\alpha^{SC}}\right) \tag{5.99}$$

Finally, it is worth mentioning that every cost in a single adaptation step can be computed as the sum of the costs endured by all assistants. For instance, the information retrieval cost associated with $SocStr$ in adaptation step x can be expressed as the sum of the costs of collecting the information for each assistant i (i.e. $c_{info_{x,i}^{SS}}$):

$$c_{info_x^{SS}} = \sum_{i=1}^{n} c_{info_{x,i}^{SS}} \tag{5.100}$$

5.5.4.3 Example

Again, we can use the traffic scenario to illustrate previous concepts. Regarding information retrieval costs ($c_{info^{O}}$), there is a cost associated to observable properties, since assistants require compiling information from radar traps (used to detect vehicle speeds, i.e. AgP) and automatic traffic counters (used to estimate road densities, i.e. $EnvP$). There is also a cost of performing the computations ($c_{comp^{O}}$) of the adaptation functions and achieving agreement between the assistants. Moreover, upon organisational component updates, domain-level agents need to be informed, and this implies an an adoption cost ($c_{adopt^{O}}$). For example, when speed limit is updated, all electronic traffic signs must be updated accordingly. Furthermore, when this new speed limit is already communicated, there is a delay ($c_{trans^{O}}$) to allow the vehicles to adapt their speeds, since they cannot

[10] Notice that for the sake of simplicity, we express adaptation frequencies as fixed time intervals. However, they can be dynamic depending on system status.

change their speed abruptly. Finally, the traffic scenario also requires goals to be updated at a lower frequency. Thus, for instance, the frequency of the norm adaptation function is lower than that of the goals (i.e. $freq_{\alpha^N} < freq_{\alpha^G}$). This allows assistants to monitor traffic flow and collisions to measure goal fulfilment after updating the speed limit. Afterwards, it will only be possible to compare current performance results with previous ones if both have been computed by considering the same reference goals.

Chapter 6

Case study: P2P sharing network

The Peer-to-Peer sharing network scenario introduced in the first chapter constitutes the case study of this thesis. Briefly, it consists of a set of computers sharing some data, so that at the end of the process all of them have a copy of it. In these networks, these computers are controlled by third parties, in such a way that it is not possible to assign tasks to each of them, but just to regulate their activity instead. Accordingly, this scenario can be regarded as a regulation-oriented problem where it is possible to use a regulation definition approach.

This chapter provides a complete scenario description and specifies an AOCMAS within this domain according to our proposed architecture (2-LAMA). The scenario description details the underlying communication network and the widely used BitTorrent protocol —which is the baseline of our proposal. Besides, the specification stipulates the actual protocol and norms in terms of our 2-LAMA model.

6.1 Introduction

The first chapter stated that we focus on those problems driven by goals that cannot be decomposed into subtasks assumed by agents —see §1.2. We called them regulation-oriented problems, and we illustrated these problems in different scenarios: auctions, traffic and peer-to-peer sharing networks —see §1.2. Moreover, we denoted as regulation definition approaches those OCMAS that can deal with such kind of problems.

Then, the first exemplifying scenario (an auction) was used in chapter 4 to illustrate a set of assistance services that we propose. Afterwards, the second scenario (traffic) illustrated our own regulation definition approach (2-LAMA) in chapter 5. Now, this third scenario (a P2P sharing network) is fully described to use it as the case study of this thesis. In other words, current chapter provides

all details about this scenario, including its specification according to 2-LAMA model —see §5.5. Next chapters use this specification to illustrate our adaptation mechanism proposals. In fact, these proposals are implemented in our sharing network simulator and tested empirically.

As introduced in section §1.2.2, a Peer-to-Peer sharing network (P2P henceforth) is composed of third-party computers that share some data. Its target is that all these computers have a copy of this data as soon as possible. Since these computers are controlled by third parties, its goal cannot be decomposed into tasks that computers assume, so the scenario is an instance of a *regulation-oriented problem*. As a result, existing P2P approaches are based on regulations over computers' activity instead of assigning tasks —i.e. they are *regulation definition approaches*. For example, the widely used BitTorrent protocol (BT) [BitTorrentInc., 2001] is just a regulative framework (specified by a protocol) that determines the social conventions among these computers.

Current P2P networks are highly complex, so we try to reduce their complexity by assuming certain simplifications about their protocol and the underlying network model. For instance, some P2P protocols require to locate initially the desired data. Notwithstanding, BT protocol do not deal with this searching phase, but delegates it to other existing mechanisms —e.g. web-based search. Accordingly, as our approach is based on BT, it neither deals with this searching phase. Nevertheless, original BT protocol splits the sharing data into several pieces, so that these pieces are exchanged in a single protocol message —i.e. the whole data requires several piece messages. In contrast, for the sake of simplicity, in our approach the data has only a *single piece*. Additionally, current P2P networks usually run on the Internet, so there is a heterogeneous mixture of network links among computers which have several distinct parameters. However, we abstract all low-level network details, and simulate a bare *packet switching network*. In fact, the low-level protocols of the Internet are based on this model.

Notice that the environment of this scenario fulfils the features described in §1.2.1. It has an *inaccessible non-deterministic dynamic* environment since the details about Internet traffic –i.e. changes in aggregated links– are unavailable, so it is not possible to predict message latencies —which may change while computers choose their fastest partner. It has a *continuous* state space and action space, for instance bandwidth consumptions are continuous. It has a *run-time adaptation* since norms are updated while agents are exchanging messages.

Next sections provide further details about the actual protocol and network in our scenario[1]. In particular, the first sections describe the base-line scenario, whereas the last sections specify our 2-LAMA approach.

6.2 BitTorrent protocol

Nowadays BitTorrent (BT) is one of the most widely used protocols in P2P sharing networks. There are several extensions to the original protocol, however our approach is based on the basic BT protocol [BitTorrentInc., 2001].

[1] There is an initial introduction to the P2P scenario in §1.2.2.3.

Protocol Phase	Roles	Protocol Messages
Initial	both	`join <hasDatum>, contact <peers>`
	peers	`handshake, bitfield <hasDatum>`
Data sharing	peers	`interested, choke, unchoke, request, data`
Notification	both	`complete`
	peers	`have`

Table 6.1: BitTorrent Protocol messages.

In basic BT there are two kinds of nodes: peers and trackers. On the one hand, a regular computer that participates by sharing data is called a *peer*. All peers share the same data. When a computer has the whole data, it is called a *seed*, whereas it is named a *leech* otherwise. In this thesis, we also refer to a seed/leech as a complete/incomplete peer respectively. On the other hand, in a basic BT network, there is a computer called *tracker* which keeps a list of computers sharing the same data —i.e. it provides a directory service. Accordingly, if other data is shared, there is a separated BT network with another tracker. Besides, as stated in previous section, our simplified version of BT works with a single-piece datum instead of multi-piece data[2].

The BT protocol defines how these nodes interact among them in order to share the data. Table 6.1 contains a list of messages involved in our simplified version of this protocol. The original BitTorrent has no phases, however we introduce them to make the protocol comprehension easier. Moreover, these phases facilitates comparing this protocol to its derivative used in 2-LAMA — which is specified in §6.7. Even more, in both protocols, we also specify the roles involved. In BT protocol, it may be only the peers (peers), or both the peers and the tracker (both). Next subsections describe each phase, and illustrate them in the example depicted in figure 6.1. In this example, there are three peers (p_i) and one tracker.

6.2.1 Initial phase

Initially, a computer gets involved in a sharing network by sending a message (`join <hasDatum>`) to the network's tracker. This message indicates if the new participant has (`<hasDatum>=1`) or has not (`<hasDatum>=0`) the datum —i.e. if it is complete or incomplete. Then, the tracker replies with a list of all current connected computers (`contact <peers>`). So, the new participant can send messages (`handshake`) to contact them. After this initial handshake, they exchange information (`bitfield <hasDatum>`) about datum possession[3].

For instance, in figure 6.1, computers p_1 and p_2 send a message (`join`) to join to the sharing network at tick t_1 and t_2. As a result, the tracker sends the

[2]Original BT protocol split large data into small pieces. Then, computers collect these pieces from different sources to compose the whole data again.

[3]In the multi-piece BitTorrent, the "`bitfield`" message contains one bit for each piece to indicate if the peer has it or not.

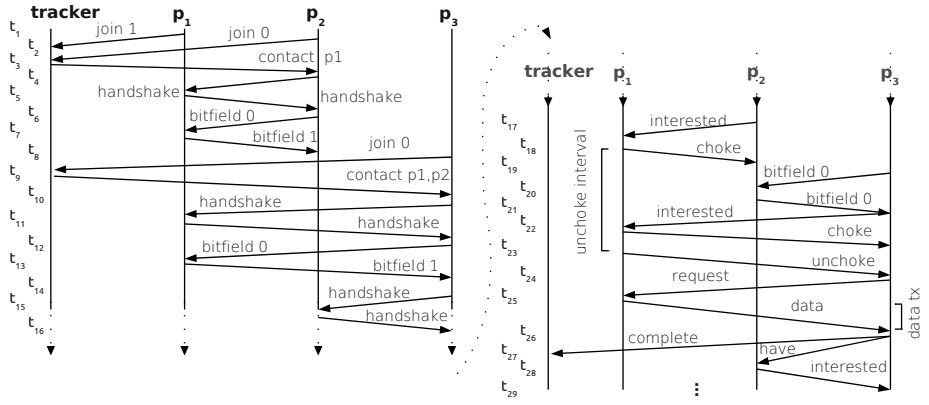


Figure 6.1: BitTorrent simplified protocol example.

list (`contact`) of current participants to p_2 at t_3 —notice that it is only necessary to send the contact to one out of a pair of contacted peers. Consequently, p_2 contacts (`handshake`) p_1 at t_4. Afterwards, both of them exchange information about their datum possession (`bitfield`) at t_6 and t_7.

6.2.2 Data sharing phase

After the initial phase, participants can start the data sharing phase with their contacts. In order to start this phase, an incomplete agent shows its interest (`interested`) to some of its complete contacts. As a response, it receives a blocking message (`choke`) meaning that any further communication will be ignored.

Nevertheless, at certain time intervals (`unchoke_interval`), each participant having the datum sends an unblocking message (`unchoke`) to some of the peers that were interested. The BitTorrent specification defines that four peers (`num_unchokes`) are selected among the interested ones (candidates). The selected candidates are those that were blocked most recently. In case two of them were blocked at the same time, the one having a larger network transmission capacity (`upload_bw`[4]) is selected. In fact, if an agent's interest is older than a defined interval (`ageing_period = unchoke_interval`/1.5), its age is ignored and only its peer's `upload_bw` is compared. In addition, in two out of three `unchoke_interval` selection processes, the fourth candidate is randomly selected. Finally, when an agent receives an unblocking message (`unchoke`) and it is still incomplete, then it asks for the datum (`request`). Upon the reception of the request, the complete peer sends the datum (`data`).

As an illustration, see the right part of figure 6.1. After the handshake, the incomplete computer p_2 sends an interest message (`interested`) to the p_1 at t_{17}.

[4]In a multi-piece scenario, this measure is estimated from previous piece interchanges. However, since in a single-piece implementation no estimation can be performed, its value is taken from the network definition.

Immediately, p_1 replies with a blocking message (`choke`) at t_{18}. Analogously, p_3 shows its interest at t_{21} and is blocked at t_{22}. Hence, p_2 and p_3 are the candidates of p_1. Then, after the `unchoke_interval`, p_1 unblocks some of its candidates. For illustrative purposes, in this example p_1 simply unblocks (`unchoke`) a single candidate: p_3 at t_{23}. When p_3 knows at t_{24} that it is unblocked, it asks for the datum (`request`) to p_1. As a result, p_1 starts sending the datum (`data`) at t_{25}.

6.2.3 Notification phase

Whenever a computer receives the whole datum, it starts the notification phase. First, it notifies the tracker (`complete`) that it has the datum. Next, it notifies (`have`) its incomplete contacts that its status has changed. Then, some of these contacts may probably show their interest as described before.

For example, in figure 6.1, after the datum has been transmitted from p_1 to p_3 during a time interval (*data tx*), p_3 starts the notification phase. Specifically, at t_{26} it notifies the tracker (`complete`) and its incomplete contact p_2 (`have`) that it has the datum. As a consequence, p_2 shows its interest (`interested`) at t_{28}.

The described simplified version of BitTorrent is the protocol used as baseline to empirically compare it to our approach. Also, it is the base for the protocol used in our approach, as described in §6.7.

6.3 Network abstraction

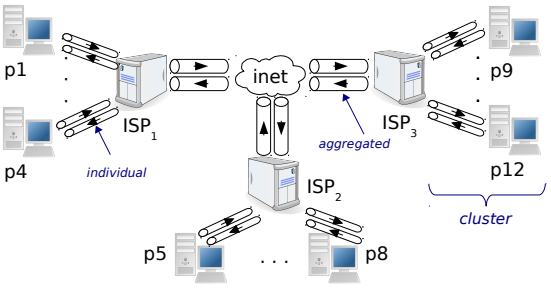
We consider the P2P underlying communication network as the environment in which the sharing process takes place. Notice that the network topology and its saturation influences the communication time of messages between agents. In our network model, we have individual and shared channels, which can be regarded as individual and shared resources —this is a common situation in many MAS scenarios.

6.3.1 Topology

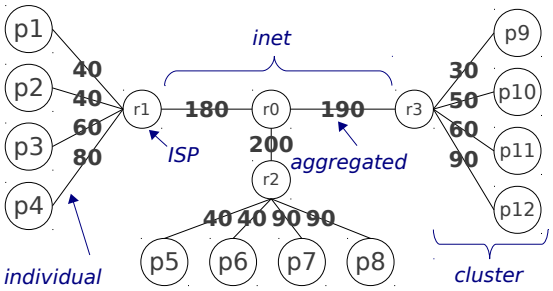
We conceptualise the network as a packet switching net having the topology illustrated in figure 6.2 —notice that fig. 6.2a is a figurative version similar to the introductory fig. 1.4, whereas fig. 6.2b is a more abstract version. There is an *individual link* between each peer (p_i) and its Internet Service Provider (ISP_i or $r_{i>0}$[5]). Besides, the Internet (*inet*) is abstracted as a set of *aggregated links* among ISPs. Each of these aggregated links transports the messages of an entire *cluster* (those peers connected to the same ISP). In all cases, links have independent upload and download channels with the bandwidth[6] represented as single numerical labels in fig. 6.2b —for simplicity, we assume both directions

[5] In figure 6.2b, r_i nodes act as routers among different links.

[6] The *bandwidth* is the capacity to transfer data over a network link, expressed as #data units per time unit —see §6.3.2.



(a) Figurative



(b) Abstract

Figure 6.2: Network abstraction.

have equal bandwidth. This means that each peer has its own communication capacity.

Notice that the time required to transmit a message from one computer to another depends on: its length, the bandwidths of the traversed links, and the number of simultaneous messages traversing the same links. As an illustration in the network depicted in figure 6.2b, we assume p_2 sends a message of 200 data units to p_9. First, such message is split into packets of a fixed size, for instance 10 units. Hence, at $t = 0$ the original message has 20 packets as illustrated in table 6.2. Given the fact that the bandwidth of p_2's individual link (ln_{r1}^{p2}) is 40, it can transmit 4 packets at each tick. So, at $t = 1$ the first four packets ($pk_{1..4}$) are injected into the network (through ln_{r1}^{p2}). Consequently, at $t = 2$ these packets ($pk_{1..4}$) achieve r_1 and are routed towards r_0 trough the aggregated link of ISP_1 (ln_{r0}^{r1}). At the same tick, the subsequent four packets ($pk_{5..8}$) are being transmitted through p_2's individual link (ln_{r1}^{p2}). Hence, at $t = 3$, ln_{r1}^{p2} is transmitting $pk_{9..12}$, ln_{r0}^{r1} has $pk_{5..8}$ and ln_{r3}^{r0} has $pk_{1..4}$. However, at $t = 4$ the individual link of p_9 only has bandwidth to transmit 3 packets instead of 4. Thus, ln_{p9}^{r3} is transmitting $pk_{1..3}$ and the fourth packet pk_4^{p2} is waiting in the buffer of router r_3. Consequently, at $t = 5$, p_9 starts receiving packets at a pace of 3 packets per tick. In other words, without any further traffic, the packet cadence among p_2 and p_9 is limited by the narrowest link —in this case

t	p2	ln^{p2}_{r1} bw=40	r1	ln^{r1}_{r0} bw=180	r0	ln^{r0}_{r3} bw=190	r3	ln^{r3}_{p9} bw=30	p9
0	$pk_{1..20}$								
1	$pk_{5..20}$	$pk_{1..4}$							
2	$pk_{9..20}$	$pk_{5..8}$		$pk_{1..4}$					
3	$pk_{12..20}$	$pk_{9..12}$		$pk_{5..8}$		$pk_{1..4}$			
4	$pk_{16..20}$	$pk_{12..16}$		$pk_{9..12}$		$pk_{5..8}$	pk_4	$pk_{1..3}$	
5		$pk_{16..20}$		$pk_{12..16}$		$pk_{9..12}$	$pk_{7..8}$	$pk_{4..6}$	$pk_{1..3}$

Table 6.2: Packet transmission example.

ln^{r3}_{p9}. Next section contains further details about simultaneous usage of a link by packets coming from different sources. Section 8.3.5 in the simulator's chapter, has more details about the implementation of the described model.

6.3.2 Metrics

We define the *bandwidth* (bw_{c_i}) as the number of data units that can traverse a channel (c_i) in a time unit. Hence, we define the *channel usage* (usg) as the ratio of the bandwidth that is used in a given time unit (t). Equation 6.1 formalises it, where $\#msg^t_i$ stands for the number of messages traversing the i-th channel in that moment t, $msg^t_{j,i}$ denotes the j-th message in the channel, and $trans$ stands for the effective data units transmitted during that period. For instance, if a channel has a bandwidth of 6 units and there is a message of 6 data units, such message traverses the link in one time unit and the channel has an usage equal to 1 during this time. Accordingly, if there are two messages of 6 data units each, both spend two time units to traverse the channel —channels transmit the same portion of every pending message. In such case, the usage is 1 during this period. In contrast, if there is only one message of 3 data units, it still traverses the channel in one time unit but its usage is 0.5 during this period.

$$usg\left(c_i, t\right) = \frac{\sum_{j=0}^{\#msgs^t_i} trans\left(msg^t_{j,i}\right)}{bw_{c_i}} \tag{6.1}$$

In other words, the time required by a message to travel from a computer to another one depends on the bandwidth and the usage of the channels it traverses. In this sense, the term *latency* denotes the time required for a message of one data unit to be transmitted among two peers. Thus, if a computer is receiving a lot of messages at the same time, the usage of its *individual* download channel is high and their latencies increase —it takes longer to deliver all messages to it. Analogously, if every computer of the same ISP are receiving a few messages, the usage of their individual download channels are low and their latency is small —i.e. messages are delivered fast. However, as all these messages traverse the *aggregated* download channel of the same ISP, the usage of this channel may be high and messages may be slowly delivered. Consequently, it may not be a good

strategy that all computers use all their communication bandwidth, because network channels may be full and it would delay the complete sharing process.

The defined usage metric does not distinguish between a fully occupied channel –which operates normally– and a channel with delayed messages due to excessive traffic. In the latter situation, message latencies increase boundlessly whilst usage cannot increase any further than 1. Therefore, we introduce another metric to identify and quantify this situation. We define *channel saturation* (sat) as the amount of data waiting to be transmitted over the amount of data the channel can actually transmit in a unit time, as formalised in equation 6.2. In such equation, $portion_{to\,trans}$ denotes the remaining data units of a message to transmit through a given channel. For instance, if a channel has a bandwidth of 6 and there is a message of 6 data units, this message traverses the link in one time unit and we say its saturation is 1 during this time. In contrast, if there is one message of 6 data units and another message of 12 data units, the saturation is 3 the first time unit ($= \frac{6+12}{6}$), 2 the second time unit ($= \frac{3+9}{6}$) and 1 the third last unit ($= \frac{0+6}{6}$). The average of this saturation is 2 ($= \frac{3+2+1}{3}$) meaning the channel had, in average, double the data than it could transmit.

$$sat\,(c_i, t) = \frac{\sum_{j=0}^{\#msgs_i^t} portion_{to\,trans}\left(msg_{j,i}^t\right)}{bw(c_i)} \tag{6.2}$$

These network metrics can be used to describe the network state at a given moment. Furthermore, their average along a whole sharing process can be used characterise the whole process. In particular, we call *network usage* ($netUsg$) to the average of the usage of all channels along a sharing process as shown in equation 6.3. In such equation, $\#ch$ stands for the number of channels in the whole network, and t_{spread} denotes the number of time steps until all peers have the data. Analogously, equation 6.4 shows the *network saturation* ($netSat$) metric.

$$netUsg = \frac{\sum_{t=0}^{t_{spread}} \frac{\sum_{i=1}^{\#ch} usg(c_i,t)}{\#ch}}{t_{spread}} \tag{6.3}$$

$$netSat = \frac{\sum_{t=0}^{t_{spread}} \frac{\sum_{i=1}^{\#ch} sat(c_i,t)}{\#ch}}{t_{spread}} \tag{6.4}$$

In addition, we also define another metric that can be use to characterise a whole sharing process. It is is proportional to the amount of data transmitted and the distance traversed by it. Specifically, we define the *cost of a message* ($cMsg_k$) as its size ($||m_k||$) times the number of links it traverses ($||path\,(m_k)\,||$) as shown in equation 6.5 —the number of traversed links is also referred as *hops*. Further, we define the *network cost* ($cNet$) as the sum of all message costs along a sharing process. Equation 6.6 shows this cost, where $\#msgs$ stands for the number of messages along all the execution.

$$c_{m_k} = ||m_k|| \cdot ||path(m_k)|| \tag{6.5}$$

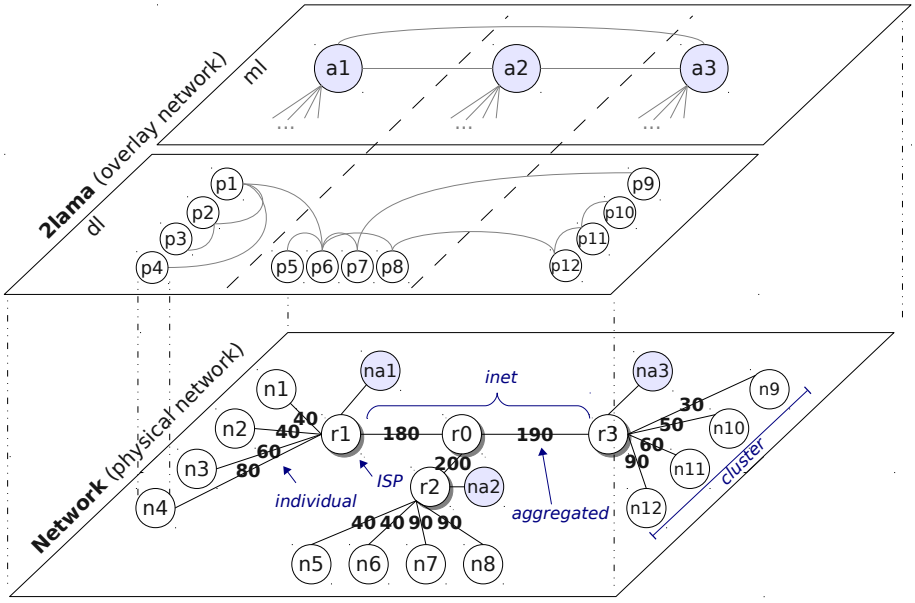


Figure 6.3: 2-LAMA over a physical network.

$$c_n = \sum_{k=1}^{\#msgs} c_{m_k} \tag{6.6}$$

Finally, in order to keep the model simple, we assume that our sharing process is the only one that generates network traffic. This implies that the observable environment properties are only affected by system's activity.

6.4 2-LAMA specification

We regard the described P2P sharing network as an OCMAS, where computers are agents interacting according to a given coordination model. We assume this coordination model can be adapted to improve system's performance. In other words, we regard the P2P scenario as an AOCMAS that can be modelled according to our 2-LAMA architecture.

In particular, figure 6.3 illustrates our AOCMAS approach of the P2P scenario in accordance with the 2-LAMA model defined in §5.5. The top part of this figure represents the agents, whereas its bottom part shows the underlying communication network described in §6.3.1. Hence, a participant computer –so-called a peer in BT– is symbolised by an agent p_i on top of a network adaptor n_i —e.g. see p_4 over n_4 in fig. 6.3. Hence, the bottom part of figure 6.3 corresponds to figure 6.2b. Notice that in the P2P scenario, the net of actually contacted

agents forms an *overlay network* since it is a net over the underlying physical network. Besides, the figure also shows assistants as agents a_i on top of network adaptors na_j which are attached to ISPs.

Specifically, the top part of figure 6.3 illustrates a *particular specification* ($2lama_{p2p}$) of the 2-LAMA model ($2LAMA$) in the P2P scenario as formalised in equation 6.7. As a result, such specification has its own meta-level ($ml_{p2p} \in ML$), domain-level ($dl_{p2p} \in DL$) and assistance functions ($\gamma_{p2p} \in AssistF$ carried out by ml_{p2p}) as expressed in equation 6.8.

$$2lama_{p2p} \in 2LAMA \tag{6.7}$$

$$2lama_{p2p} = (ml_{p2p}, dl_{p2p}, \gamma_{p2p}) \tag{6.8}$$

Next sections provide further details on these components. For the sake of simplicity, in these sections we omit the 'p2p' subscript in equations since all of them refer to the P2P scenario. Also, we omit indicating that each symbol in lower-case letters belongs to the corresponding upper-case symbol defined by the model —like $dl \in DL$.

6.5 Domain-level specification

In the P2P scenario, the domain-level is the one that performs the sharing activity. According to our 2-LAMA model, it has its own set of agents (ags_{DL}) which interact within an environment (env_{DL}) according to an organisation entity (org_{DL}) —see eq. 6.9.

$$dl = (ags_{DL}, org_{DL}, env_{DL}) \tag{6.9}$$

$$org_{DL} = (socStr_{DL}, socConv_{DL}, goals_{DL}) \tag{6.10}$$

Therefore, domain-level agents correspond to the computers that share the datum. For instance, in the example illustrated in figure 6.3, as there are 12 peers, the execution state would have $ags_{DL} = \{p_1, \ldots, p_{12}\}$. Hence, the underlying physical network described in §6.3 is part of their environment. For instance, the network illustrated in the bottom part of figure 6.3. Finally, their organisation is based on the BitTorrent protocol detailed in §6.2. Next subsections specify the observable properties of the first two elements ($AgsP_{DL}$, $EnvP_{DL}$) and the three organisational components ($socStr_{DL}$, $socConv_{DL}$, $goals_{DL}$) —see eq. 6.10 and §5.3. Notice that in the subsequent equations we omit the 'DL' subscript since all of them refer to the domain-level.

6.5.1 Observable properties

The organisation specification refers to the observable properties of agents and environment instead of themselves —see §5.3. Furthermore, the observable properties are essentially compounds of information which are local to clusters —see §5.5.1.2. Accordingly, $2lama_{p2p}$ specifies the properties of a single

agent (AgP), the properties of the environment of a cluster ($EnvP^{C}$) and those properties that are not associated to any particular cluster ($EnvP^{SnC}$). These properties are described as follows and are used later when specifying the organisational components.

6.5.1.1 Agent Observable Properties

The observable properties of a single agent are related to datum possession and agent's activity as defined by equations 6.11-6.16.

$$AgP = Has \times Compl \times Act \times Pace \times SimultDataMsgs \tag{6.11}$$

$$Has = \{\texttt{yes}, \texttt{no}\} \tag{6.12}$$

$$Compl = \{n : n \in \mathbb{R} \wedge n \leq 100\} \tag{6.13}$$

$$Act = \{\texttt{none}, \texttt{receiving}, \texttt{serving}\} \tag{6.14}$$

$$Pace = \{n : n \in \mathbb{R} \wedge n \leq 100\} \tag{6.15}$$

$$SimultDataMsgs = \mathbb{N} \tag{6.16}$$

On the one hand, the datum possession is described by two properties: Has and $Compl$. The former (Has) is simply a boolean property that indicates if the agent has (`yes`) or has not (`no`) the shared data. At the same time, the latter property ($Compl$) provides more detail since it expresses the percentage of data possession. Its value ranges from 100% (if the agent has the datum) to 0% (if it has not the datum). Intermediate values describe situations where an agent already has received some packets of the corresponding data message.

On the other hand, the agent's activity is described by three properties: Act, $Pace$, $SimultDataMsgs$. The former (Act) is a general description of the activity which details if the agent is receiving the datum (`receiving`), serving it (`serving`) or inactive (`none`). The second property ($Pace$) is a measure of the pace with which an agent is transmitting data —i.e. an upload ratio. This measure is the percentage of nominal bandwidth that the agent is using to send data in the current time instant —all protocol message types are included. For instance, if an agent sends a message of 20 data units through its upload channel of bandwidth 5 without any restriction, its $Pace$ is 100% and will require 4 time units to be transmitted. However, if the agent decides to send it at a $Pace$ of 40% (i.e. 2 units), it will require 10 time units.

Finally, the latter property ($SimultDataMsgs$) is a measure of the amount of data messages that are sent simultaneously in the current time instant. As an illustration, we assume that an agent with a nominal bandwidth of 10 starts sending a data message at tick 1 and another one at tick 3. If those messages require 100 data units to be transmitted, then at tick 4 the $SimultDataMsgs$ is 2 since the agent is sending both messages simultaneously.

6.5.1.2 Environment Observable Properties

The observable properties of the environment are related to the time consumed to spread the datum and the network bandwidths used by each participant. The time consumed is not associated to any particular cluster ($EnvP^{SnC}$), whereas the bandwidths are related to the environment of each cluster of peers ($EnvP^{C}$).

Specifically, $EnvP^{SnC}$ is the sharing time ($ShTime$) as expressed in equations 6.17-6.18. It is worth to mention that in subsequent chapters, we may also refer to this time as t_{spread}.

$$EnvP^{SnC} = ShTime \tag{6.17}$$

$$ShTime = \mathbb{N} \cup \{\texttt{undef}\} \tag{6.18}$$

This sharing time ($ShTime$) is the number of ticks from the beginning of the data sharing process up to the moment in which all computers have the datum. Since this sharing time is unknown until the end of the process, this property has an undefined value (`undef`) until then. As a result, the meta-level can use this property to evaluate a whole sharing process and compare it with previous ones. Nevertheless, it cannot use this property to evaluate system's evolution during the sharing process —see §6.5.4 for a further discussion on this topic.

On the other hand, the information related to a cluster ($EnvP^{C}$) is composed by: a detailed information about the bandwidth of its agents ($NetBW^{C}$), the measured communication latencies among participants ($NetLat^{C}$) and a metric that evaluates their degree of connectivity ($Connect^{C}$) —see eqs. 6.19-6.23.

$$EnvP^{C} = NetBW^{C} \times NetLat^{C} \times Connect^{C} \tag{6.19}$$

$$NetBW^{C} = \{netBW_i : netBW_i \in NetBW \wedge (1 \leq i \leq m)\} \tag{6.20}$$

$$\begin{aligned} NetBW \quad = \quad & NomBW \times EffUpBW \times EffDnBW \\ & NomBW = \mathbb{N},\ EffUpBW = \mathbb{R},\ EffDnBW = \mathbb{R} \end{aligned} \tag{6.21}$$

$$\begin{aligned} NetLat^{C} \quad &= (NetLat)^{m\cdot(m-1)} \\ NetLat \quad &= n, \quad n \in \mathbb{R} \wedge n \geq 0 \end{aligned} \tag{6.22}$$

$$\begin{aligned} Connect^{C} \quad &= (Connect)^{m} \\ Connect \quad &= n, \quad n \in \mathbb{R} \wedge n \geq 0 \end{aligned} \tag{6.23}$$

In particular, the information about the bandwidth of participants ($NetBW^{C}$) contains detailed information about the bandwidth of each i-th agent in the cluster ($NetBW$) —in eq. 6.19, m stands for the number of agents in a cluster. This information includes the nominal bandwidth ($NomBW$), which is the physical maximum available bandwidth of an agent's individual link —it is the same for both the upload and download channels, see 6.3.1. Also, it includes the agent's upload effective bandwidth ($EffUpBW$), which is its current actual bandwidth consumption used to send data. For instance, a participant may have an individual link with a bandwidth of 20 units –i.e. $NomBW = 20$– but it may

be sending only 15 data units in the current time instant —i.e. $EffUpBW = 15$. Similarly, it also includes agent's download effective bandwidth ($EffDnBW$), which is its current actual bandwidth consumed to receive data. For example, due to a network saturation in some aggregated links, an agent may be receiving only 10 units of data –i.e. $EffDnBW = 10$– instead of the 20 units allowed by its nominal bandwidth.

Besides, the communication latencies ($NetLat^C$) and the connectivity metric ($Connect^C$) are described in §7.2.1.1 when discussing about the social relationships adaptation. Briefly, the communication latencies ($NetLat^C$ in eq. 6.22) are measures of the communication delay among each pair of agents in a cluster —i.e. there are $m \cdot (m-1)$ measures. Whereas the connectivity metric ($Connect^C$, eq. 6.23) is a single value for each cluster participant that summarises all its communication latencies —i.e. there are m values.

6.5.2 Social Structure

According to 2-LAMA, the social structure specification ($socStr$) has three components: roles, groups and relationships ($rels$) —see §5.3.1. Equations 6.24-6.27 specify them in the P2P scenario.

$$socStr = (roles, groups, rels) \tag{6.24}$$

$$roles = \{\texttt{peer}\} \tag{6.25}$$

$$groups = \{\texttt{cluster}\} \tag{6.26}$$

$$rels = \left\{Contact(ag_i, ag_j) : \texttt{peer} \in \mathtt{R}_{ag_i} \wedge \texttt{peer} \in \mathtt{R}_{ag_j}\right\} \tag{6.27}$$

Due to the flat organisation of BitTorrent protocol (a pure peer-to-peer organisation), there is a single role (`peer`) played by all the computers that share the datum[7]. Also, there is a single sort of grouping structure (`cluster`), which refers to those participants connected to a same ISP. Moreover, there is a single relationship among peers ($Contact$), which exists between two participants if they know about each other. Hence, these relationships constitute the overlay network over the physical network as illustrated in figure 6.3.

6.5.3 Social Conventions

The social conventions have three components in 2-LAMA: protocols ($prots$), norms ($norms$) and a detection policy ($detecPol$) —see §5.3.2. The equations 6.28-6.31 specify such components in the P2P scenario.

$$socConv = (prots, norms, detecPol) \tag{6.28}$$

$$prots = \{bitTorrent'\} \tag{6.29}$$

[7]Notice that the tracker role –defined by BitTorrent protocol in §6.2– belongs to the Organisational Layer in our Coordination Support perspective. Hence, it does not belong to domain-level participant agents, as discussed in §6.7.

$$norms = \{normBW, normFR\} \tag{6.30}$$

$$detecPol = \texttt{full} \tag{6.31}$$

There is a single protocol ($bitTorrent'$) which is a derivative of the BitTorrent protocol described in §6.2.[8] We regard the original BitTorrent as a non-adaptive coordination model, since its coordination mechanisms are not altered along the execution. In contrast, we propose an adaptive coordination model, which includes two norms that are adapted by the meta-level. These norms differ in the number of involved interactions. On the one hand, there is a single-interaction norm ($normBW$) that limits the amount of upload bandwidth that agents can use. This way, participants cannot use the network as an infinite resource. On the other hand, we define a multiple-interaction norm ($normFR$) that limits the number of simultaneous data messages a computer can send. In this manner, meta-level can indirectly regulate the amount of simultaneous data transmissions to avoid network channel saturation.

In this specification, the violations of such conventions are fully detected by a mechanism located at ISPs which filters out offending messages. This approach is not unrealistic since, nowadays, there exist ISP initiatives [Xie et al., 2008] to improve –and be involved in– P2P distribution systems. Accordingly, the detection policy specifies there is a full detection (`full`) of convention violations.

Norms

As mentioned, the $normBW$ prevents agents from making massive use of their bandwidth to send/receive data to/from other participants. It declares that an agent cannot use more than max_{BW} percentage of its nominal bandwidth as formalised in eq. 6.32. In particular, it prohibits (prh) that a domain-level agent (ag_i) uploads to the network ($pace_{ag_i} \in Pace$) more than a given threshold (max_{BW}) of data units at a certain moment —see §6.5.1.1 for further details about $Pace$ agent property.

$$normBW = \mathsf{prh}\ (\texttt{peer} \in \mathtt{R}_{ag_i} \wedge pace_{ag_i} > max_{BW}) \tag{6.32}$$

Limiting the network usage may increase the execution time since messages are sent at a lower pace. However, using it as an infinite resource can also increase the execution time because network channels become saturated and message latencies increase. Thus, depending on network status, different max_{BW} can be useful in order to use as much network as possible without saturating it. Hence, the value of max_{BW} is not defined by the designer at the specification stage, but it is defined at run-time by the meta-level.

In contrast, the $normFR$ is targeted to limit the number of simultaneous data messages. It defines that a participant cannot send simultaneously the datum to more than max_{FR} peers. In other words, it limits the number of

[8] Such derivative is detailed later in §6.7, since it also contains messages related to the meta-level which is specified in next section.

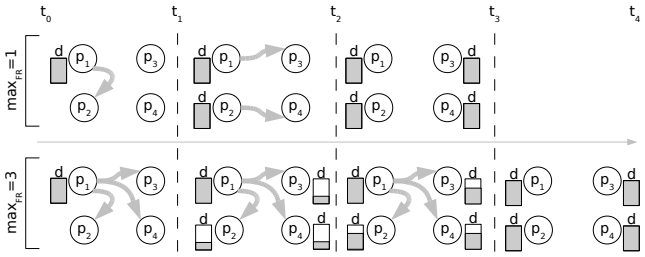


Figure 6.4: *normFR* example (*d* stands for datum, and *t* for time).

receiving agents (we call them *friends* of the sending agent) to whom a participant can simultaneously send the data. It is formalised in equation 6.33, where there is a prohibition (prh) of a formula that is true if a domain-level agent (ag_i) is simultaneously sending ($simultDataMsgs_{a_i} \in SimultDataMsgs$) more than max_{FR} data messages at a certain moment —see §6.5.1.1 for further details about $SimultDataMsgs$ agent property.

$$normFR = \mathsf{prh}\ (\mathtt{peer} \in \mathtt{R}_{ag_i} \wedge simultDataMsgs_{ag_i} > max_{FR}) \tag{6.33}$$

In this manner, the meta-level can also alter the network traffic by concentrating (small max_{FR}) or spreading (large max_{FR}) the data messages towards a few or a lot of destination agents. As an illustration, in figure 6.4 we assume there is an agent with the datum (p_1) and three agents without it ($p_2..p_4$). Also, we assume that all participants have the same capacity to send/receive data. Accordingly, the capacity to send data of p_1 is smaller than the capacity of receiving data of $p_2..p_4$ altogether. In such a case, if $max_{FR} = 1$, at tick t_0 agent p_1 sends the datum to only one of the other participants, which is p_2 in the figure. Then, p_2 has the datum sooner (at t_1) than if $max_{FR} = 3$ (at t_3). Because if $max_{FR} = 3$, at t_0 agent p_1 sends the datum to all participants simultaneously, and data travels slower when traversing shared links. This is depicted at t_1, because when $max_{FR} = 3$, p_2 has only part of the datum. Consequently, if $max_{FR} = 1$, at t_1 agent p_2 can start serving the datum to one of the remaining participants while p_1 serves to the other. Whereas, if $max_{FR} = 3$, at t_1 agent p_2 can not act as a source yet because it does not have the complete datum. Overall, all participants have the datum in less time when $max_{FR} = 1$ (at t_2) instead of $max_{FR} = 3$ (at t_3). However, notice that the best value for max_{FR} depends on network topology –i.e. which links are shared– and may change along the execution depending on participants –i.e. who has the datum– and network status —i.e. link saturation. As a result, like max_{BW}, the value of max_{FR} is also defined by the meta-level at run-time.

6.5.4 Goals

In 2-LAMA, organisational goals are specified by: a set of relevant observable properties (*propGoals*) and an objective function (ω) —see §5.3.3. Their specification in the P2P scenario is formalised in equations 6.34-6.38.

$$goals = (propGoals, \omega) \tag{6.34}$$

$$propGoals = \{shTime, compl\} \tag{6.35}$$

$$\omega(propGoals, shTime, compl) = \begin{cases} \gamma_{ShTime}(shTime) & \text{if } shTime \neq \texttt{undef} \\ \gamma_{Compl}(compl) & \text{otherwise} \end{cases} \tag{6.36}$$

$$\gamma_{ShTime}\,(shTime) = {}^{(max_{ShTime} - shTime)}/_{max_{ShTime}} \tag{6.37}$$

$$\gamma_{Compl}\,(compl) = {}^{compl}/_{100} \tag{6.38}$$

Since the target is to share the datum as fast as possible, the relevant observable properties contain the sharing time (*ShTime*) —as explained later, another property is required when the final sharing time is still not available. Accordingly, the objective function (ω) depends on this property, which is formalised in a separate function (γ_{ShTime}) for simplicity. This other function would have its maximum satisfaction (1) if the actual sharing time has this ideal value, i.e. zero. Otherwise, it decreases its value up to the minimum satisfaction (0) if the sharing time is too large —i.e. when it approaches the constant max_{ShTime}, which is also the instant when the simulator cancels the sharing process.

However, the sharing time is undefined until the end of the sharing process —see §6.5.1. Hence, it can only be used to compute the goal satisfaction at the end of the process, but not during this process. In other words, this property is useful to evaluate a sharing process in order to compare it with previous sharing processes —see §7.5. But it cannot be used along a sharing process to reason about the current organisational adaptation.

Alternatively, another function (γ_{Compl}) is used when the sharing time is not available (`undef`). This alternative function is based on the amount of data spread along the system, such information is available along the whole sharing process. It can be computed as an average of the degree of completeness of each agent (*Compl*, eq. 6.13). For instance, if at the beginning there is only a single agent out of 10 that has the data, then the system's completeness is 10% ($\frac{100 \cdot 1 + 0 \cdot 9}{10}$). Later, if this source is transmitting a data message to a destination agent, and the half of this message has achieved its destination, the completeness is 15% ($\frac{100 \cdot 1 + 50 \cdot 1 + 0 \cdot 8}{10}$). Finally, when all participants have the whole datum, the completeness is 100%. Notice that system completeness is somehow inversely related to the remaining time since the larger is the completeness, the shorter is expected to be the remaining time.

In particular, there is an observable property (*Compl*) that measures the described completeness. Its ideal value is 100% since it means that all peers have the datum. Hence, the corresponding function (γ_{Compl}) has a level of satisfaction that approaches its maximum (1) as the agents get the datum.

6.6 Meta-level specification

In our 2-LAMA model, the meta-level is devoted to provide assistance services to domain-level agents. However, the basic BitTorrent protocol does not contain these services. Instead, it only has a directory service provided by the tracker —see §6.2. According to our Coordination Support model, such a service is formally provided by the Organisational Layer —see §2.4.2. Therefore, the baseline scenario does not have assistance services, so it does not have the adaptation service. In fact, it has a static coordination model —as stated in §6.1.

In contrast, the 2-LAMA specification has a meta-level to update the domain-level organisation —i.e. it has an adaptive coordination model. Notice that, although domain-level is inspired in the pure peer-to-peer organisation of BitTorrent, 2-LAMA applied to this scenario is no longer following such an organisation.

In particular, the meta-level has the components specified in equation 6.39. Like the domain-level, it has three components —see figure 5.3 in the formalisation chapter. In brief, it is composed by assistant agents (ags_{ML}) which interact within an environment (env_{ML}) that let them access domain-level —see eq. 5.43.

$$ml = (ags_{ML}, org_{ML}, env_{ML}) \tag{6.39}$$

$$org_{ML} = (socStr_{ML}, socConv_{ML}, goals_{ML}) \tag{6.40}$$

Also, its organisation has the three components as shown in equation 6.40. These components are specified in the following subsections, where equations omit the 'ML' subscript since all of them refer to the meta-level. Notice that, due to the extension of the specification of the adaptation functions applied by the meta-level, they are fully specified in next chapter.

6.6.1 Social Structure

The three social structure components (roles, groups and relationships, see §5.3.1) are specified in equations 6.41-6.44.

$$socStr = (roles, groups, rels) \tag{6.41}$$

$$roles = \{\texttt{assistant}\} \tag{6.42}$$

$$groups = \{\texttt{cluster}\} \tag{6.43}$$

$$rels = \left\{ ContactA(ag_i, ag_j) : \texttt{assistant} \in \mathtt{R}_{ag_i} \wedge \texttt{assistant} \in \mathtt{R}_{ag_j} \right\} \tag{6.44}$$

Essentially, there is a single role in the meta-level, which is `assistant`. The agents that play this role (i.e. all meta-level agents) are system trusted agents (i.e. staff agents) that can reason at a higher level of abstraction than peers —see §5.4 and §5.5.1.3. Each assistant provides assistance services to a disjoint subset of peers (i.e. a cluster). Hence, meta-level has also the concept of `cluster` grouping, which only includes an assistant and is aligned with domain-level clusters —i.e. it is connected to the same ISP. As an illustration, the

dashed lines in the top part of figure 6.3 depict how domain-level and meta-level are grouped into clusters. Moreover, the bottom part of the same figure shows that assistants are located at ISPs, and thus their communications are fast.

The assistant of each cluster contacts other assistants in order to exchange summaries about the information they collect —see §5.5.3.1. Accordingly, the specification includes a relationship among assistants ($ContactA$). As an illustration, these relationships are drawn as grey lines among assistants in the top part of figure 6.3.

6.6.2 Social Conventions

The social conventions are defined by the three components (protocols, norms and a detection policy, see §5.3.2) specified in equations 6.45-6.48.

$$socConv = (prots, norms, detecPol) \tag{6.45}$$

$$prots = \{bitTorrent'\} \tag{6.46}$$

$$norms = \{normHAS\} \tag{6.47}$$

$$detecPol = \texttt{none} \tag{6.48}$$

Like domain-level agents, assistants also use a derivative ($bitTorrent'$) of the BitTorrent protocol which is detailed later in §6.7. In addition, the meta-level has a norm ($normHAS$) which limits the number of domain-level agents that can be informed about new data sources in other clusters. Such norm could be adapted by an additional meta-level on top of current one as suggested in definition 6. However, in current specification, there is only one meta-level, so this norm is fixed.

Besides, as mentioned in previous subsection, assistants are staff agents. Accordingly, they are system trusted agents that do not violate neither the protocol nor the norm. Hence, the detection policy of meta-level specifies no convention violation detection (`none`).

In particular, the $normHAS$ limits the number of domain-level agents (in a cluster) an assistant can inform about a participant (in another cluster) having the data. This limit is related to the number of relationships among agents of different clusters that an assistant can suggest. In this manner, this norm regulates how assistants can update domain-level's social structure.

This norm is formalised in equation 6.49 as a prohibition (prh) that an assistant (a_i) sends more than max_{HAS} messages to a domain-level agent (p_j) to indicate that another participant has the datum. Such messages are informally called "has messages" —see §6.7. In fact, assistants have a related observable property ($HasMsgs$) which is specified in equations 6.50-6.52. It is a counter of the number of such messages sent to each participant ($HasMsgsForPeer$).

$$normHAS = \mathsf{prh}\left(\exists a_i | \texttt{assistant} \in \mathtt{R}_{a_i} \wedge hasMsgsForPeer_{a_i,p_j} > max_{HAS}\right) \tag{6.49}$$

$$AgP = HasMsgs \tag{6.50}$$

$$HasMsgs = (HasMsgsForPeer)^{|Ags|} \tag{6.51}$$

$$HasMsgsForPeer = \mathbb{N} \tag{6.52}$$

In brief, an assistant cannot simply suggest all its domain-level agents – the ones in its cluster– to contact a participant having the datum. Instead, it may have to choose only some of them. Therefore, it regulates the number of neighbours among different clusters. A small max_{HAS} reduces *aggregated* channels usage but may increase the sharing time since some agents in a cluster have to wait until the other ones receive the datum.

6.6.3 Goals and Adaptation

The meta-level goals are the same than the domain-level ones as introduced in §5.5.1.1 and formalised in equation 6.53. Accordingly, they are specified by the equations 6.34-6.36.

$$goals_{ML} = goals_{DL} \tag{6.53}$$

In order to check goal fulfilment, assistants access domain-level observable properties during the adaptation process. This is feasible since meta-level environment includes such properties —see eq. 5.43. In particular, as each assistant is in charge of a cluster of domain-level agents, it only perceives the observable properties of that group of agents. However, as explained in §5.5.3.1, an assistant receives a summary of those properties in other clusters from other assistants. The structure of this summary is specified in next chapter 7, since it is entirely devoted to organisational adaptation —see §7.3.1.

Besides, as this thesis focuses on the organisational adaptation service, in the case study we only specify some of the assistance functions related to this service. In particular, we specify the partial adaptation functions of the relationships (α_i^{Rels}) and norms (α_i^N), and their corresponding agreement functions ($\beta_{\boldsymbol{\alpha}^{Rels}}$ and $\beta_{\boldsymbol{\alpha}^N}$) in equations 6.54-6.57 —see §§5.5.2-5.5.3 about the model.

$$\alpha_i^{Rels} = optimiseLatencies \tag{6.54}$$

$$\beta_{\boldsymbol{\alpha}^{Rels}} = \cup \tag{6.55}$$

$$\alpha_i^N \in \{heuristic, cbr\} \tag{6.56}$$

$$\beta_{\boldsymbol{\alpha}^N} = voting \tag{6.57}$$

On the one hand, the adaptation of the relationships (α_i^{Rels}) is based on optimising the latencies among domain-level agents. First, an assistant obtains information about the latency of communications among its domain-level agents —see the latency metric definition in §6.3.2. Next, it reasons about such measures in order to build an overlay network of low latency paths for the transmission of the datum. Then, it sends participants the list of other agents to contact. As a result, the agreement process among assistants ($\beta_{\boldsymbol{\alpha}^{Rels}}$) is the union of their

proposed relationships, since their creation is local to each cluster. Thus, each assistant decides which updates are applied to the relationships initiated by the participants in its cluster.

On the other hand, we specify two alternatives to perform the norm adaptation (α_i^N). In can be based on an adaptation knowledge gained either at design time –by coding a heuristic– or at run-time —by using machine learning. The former, using a heuristic (*heuristic*), requires previous expert knowledge and provides a fixed estimation. Whereas the latter, using machine learning, estimates this relationship automatically and is able to evolve it along time. In particular, the machine learning technique we use is based on Case-Based Reasoning (*cbr*) [Riesbeck and Schank, 1989].

Besides, in despite of the alternative, assistants follow a voting scheme (*voting*) to achieve an agreement on how to actually update norms ($\beta_{\boldsymbol{\alpha}^N}$). In current implementation, a simple replicated averaging approach is used: each assistant computes the average for each suggested norm parameter value and communicates it to its cluster of participants.

Next chapter 7 is devoted to specify these adaptation functions. More specifically, it provides several details and examples about the relationship adaptation in §7.2 and the two norm adaptation alternatives in §7.3.

6.7 Protocol specification

Previous sections specify domain-level and meta-level components. In both cases, the protocol component contains a particular BitTorrent protocol derivative ($bitTorrent'$). This section specifies this protocol, which involves both levels. Like the base-line BitTorrent protocol defined in §6.2, its derivative is specified by a table of messages, an explanation of phases and an illustrative example.

This derivative protocol is extended to include messages between levels, as well as messages between meta-level agents. Table 6.3 presents these messages in different protocol phases. This way, it is easier to compare it with original protocol in table 6.1. Notice that there are two new phases: the social structure phase and the norms phase. These phases are related to the two organisational components that are adapted. The former (social structure phase) is devoted to estimate network latencies and vary the relationships in order to exploit meta-level assistance. Whereas the latter (norms phase) is devoted to agree new norms among assistants and communicate them to participant agents.

Next subsections detail the usage of these messages structured into protocol phases. These sections complement their specification with fragments of the example depicted in figure 6.5. This example has three agents playing the role peer (p_i) grouped into two clusters. Accordingly, there are two assistants (a_i).

6.7.1 Initial phase

In the original BitTorrent protocol, participant agents initially contact the tracker. This tracker provides a directory service as explained in §6.2. Briefly,

Protocol Phase	Levels	Protocol Messages
Initial	both	`join <hasDatum>`
Social structure	DL	`lat_req, lat_rpl, bitfield <hasDatum>`
	both	`get_lat <peers>, lat <peer> <measure>, contact <peers>`
Data sharing	DL	`request, data, cancel, choke, unchoke`
Notification	DL	`have`
	both	`complete, has_datum <peer>`
	ML	`all_complete, complete_peer <peer>`
Norms	both	`norm <norm_id> <definition>`
	ML	`vote <param> <value>, summary <sump>`

Table 6.3: 2-LAMA Protocol messages.

agents join this directory and obtain the references to other participants from it —i.e. agents use this service to be involved in their organisation. From our Coordination Support perspective, such a directory service is formally provided by the Organisational Layer —see §2.4.2. On top, the Assistance Layer can access the list of organisation's participants and update their net of relationships by interacting with the Organisational Layer. However, for the sake of simplicity, current protocol specification lets domain-level agents directly provide joining information to assistants. And it lets these assistants update agents' relationships by informing sharing computers about other participants —see next phase.

Accordingly, in the initial phase of 2-LAMA protocol, agents join a cluster by contacting the assistant in charge of it. In our specification, agents entering the system contact the closest assistant, i.e. the one they have a smallest latency with. Like in BT protocol, a new participant sends a join message (`join <hasDatum>`) which details if it has (`<hasDatum>=1`) or has not (`<hasDatum>=0`) the datum. For example, in figure 6.5 participants p_1 and p_2 send a join message to a_1 at tick t_1. These messages express that the first one (p_1) has the datum, whereas the last one (p_2) lacks it. For the sake of simplicity, figure 6.5 contains only some of the protocol messages, and the initial phase of the second cluster is omitted.

6.7.2 Social structure phase

After the initial phase, there is another phase related to the social structure adaptation (α^{Rels}). Since it is based on communication delays (eq. 6.54), assistants collect information about latencies between participants in their cluster in order to compute the lowest latency net of relationships. Hence, they start requesting domain-level agents to measure their latency with all other participants in their cluster (`get_lat <peers>`). After receiving that message agents measure their latencies with others in their cluster by sending a test message (`lat_req`, `lat_rpl`), and send a results message to its assistant (`lat <peer> <measure>`). Assistants use this information to compute a net of relationships among domain-level agents

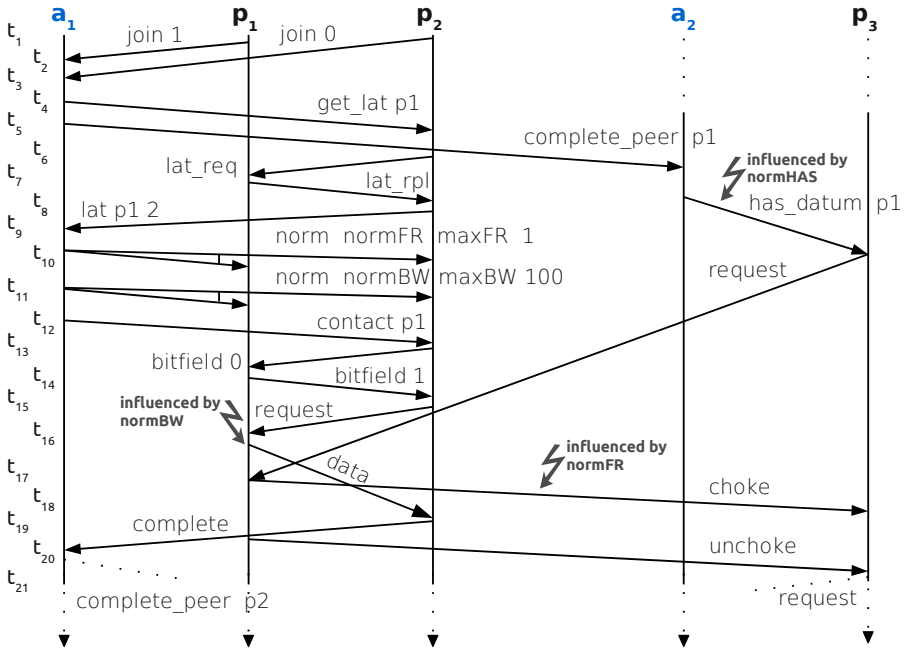


Figure 6.5: 2-LAMA protocol example.

in their clusters (see §7.2), and tell each agent which other participants it has to contact (`contact <peers>`). The actual social structure –i.e. the net of relationships that fulfil social structure specification– defines which other participants each agent can contact in order to obtain the data. After receiving their contacts, participants send a greeting message (`bitfield <hasDatum>`) to each one of their contacts, specifying whether they have the datum (`<hasDatum>=1`) or not (`<hasDatum>=0`) —i.e. the `handshake` message is omitted.

Similarly, in case a new agent enters the system, its assistant asks this *single* participant to measure its latencies against the rest of the cluster, and computes the best organisation again. On the contrary, if an agent leaves the system, its assistant can compute the new organisation without collecting new latencies. Notice that, in contrast to the BT protocol, in any case, the supplied list of contacts does not include all participants, but only a subset of agents in its cluster.

As an illustration, in figure 6.5 assistant a_1 asks p_2 to measure its latency to p_1 at t_4 (`get_lat p1`). Then, p_2 sends a latency request message (`lat_req`)at t_6 and receives its replay from p_1 (`lat_rpl`) at t_8. Later, assistant a_1 receives the corresponding elapsed time measure at t_9. In such moment, the assistant computes the net of relationships among participants. However, before sending the contact messages, it informs them about current norms as explained in §6.7.5. Then, a_1 only needs to send a contact message to one of the agents that should

contact another one. In this case, it sends the contact message (`contact p1`) to p_2 at t_{12}. As a consequence, both participants exchange the bitfield messages (`bitfield`) between t_{13} and t_{15}.

6.7.3 Data sharing phase

Once the social structure phase is finished, agents lacking the datum start a sharing phase requesting the data (`request`) from their complete contacts[9]. Notice that participants can only serve a maximum number of agents defined by the max_{FR} value —see $normFR$ in §6.5.3. Hence, upon a request, agents having the datum start sending it (`data`) if they are serving fewer participants than allowed number. Otherwise, if an agent is already serving the maximum number, it replies with a blocking message (`choke`) which denotes it can not serve the datum and that it will ignore any further messages. However, as soon as one of the current data transmissions ends, the agent informs waiting peers with an unblocking message (`unchoke`) that its status has changed. Thus, participants still lacking the datum can request it again. Agents lacking the datum are allowed to obtain data from two sources simultaneously for a short period of time. This is done in order to compare their effective bandwidth, and to choose the faster source and discard the other one by means of a cancel message (`cancel`).

For instance, in figure 6.5 the agent p_2 asks p_1 for the datum (`request`) at t_{15}. As a result, p_1 starts sending the datum (`data`) to p_2 at t_{16}. In contrast, p_3 also requests data at t_{11} but p_1 blocks it (`choke`) at t_{17}. The reason that p_1 blocks p_3 is that when it receives p_3's request at t_{17}, it is already sending data to the maximum number of agents allowed by max_{FR} —current max_{FR} value is 1. Later, as soon as p_1 finishes sending data to p_2 at t_{19}, it tells p_3 that it is available (`unchoke`). So, p_3 can request the datum again.

6.7.4 Notification phase

The notification phase is related to notify participants and assistants about new complete agents. For instance, as soon as one agent has the datum, it informs with a datum possession message (`have`) those contacts that still lack it, so they can request the datum. Furthermore, upon data reception, a participant also informs its assistant with a datum possession message (`complete`), and its assistant shares this information with other assistants with the corresponding possession messages (`complete_peer <peer>`). Afterwards, other assistants can inform some participants in their cluster about the new data source with an additional possession messages (`has_datum <peer>`). The number of agents informed depends on the meta-level norm $normHAS$ —since it defines the maximum number (max_{HAS}) of informed participants within a cluster, see §6.6.2. In that moment, informed participants may request the datum to the new source.

[9] As opposed to BitTorrent protocol, in 2-LAMA protocol there are neither interested messages nor automatic blocking replies (`interested` and `choke`). Instead, request messages (`request`) are used and the data can be just sent.

Besides, an assistant also informs other assistants when all agents in its cluster are complete with a cluster possession message (`all_complete`), preventing further unnecessary communications.

In fact, an assistant also notifies other assistants about a new complete agent (`complete_peer <peer>`) when a new complete participant joins the system. For example, in figure 6.5, when the complete agent p_1 joins assistant a_1, this one notifies a_2 that p_1 is complete (`complete_peer p1`) at t_5. Then, a_2 notifies one agent in its cluster (because $max_{HAS} = 1$) that p_1 is complete (`has_datum p1`) at t_8. This notification about a new complete agent (`has_datum`) has more semantics than a contact message (`contact`) since the former implies that the referred agent has the datum. Hence, the receiver can ask the new source for the data (`request`) without exchanging possession messages (`bitfield`) to obtain other's datum possession. This saves the time and network resources consumed by the corresponding handshake messages. Following the example in figure 6.5, at t_{12} agent p_3 is not sending a possession message (`bitfield`) to p_1, but requesting the datum (`request`).

Notice that when an assistant selects which agents are informed about a new data source (`has_datum <peer>`), it is updating the net of relationships among participants. In other words, this message is also related with the social structure adaptation (α^{Rels}). In fact, as the social structure adaptation function is related to communication latencies, an assistant chooses those incomplete agents with a lower latency as detailed in §7.2.

Furthermore, the organisational adaptation may require additional information, that is collected by assistants at the beginning of an adaptation process. For example, current norm adaptation process requires information about the degree of completeness of each agent (e.g. if it has already received 50% of the datum) —see §7.2.1.2. Hence, when an adaptation step starts, each assistant sends a message to query their participants about their degree of completeness (`provide_completeness`). As a response, each participant sends back to its assistant a message that details the percentage of the datum it has (`completeness <percentage>`).

6.7.5 Norms phase

In addition to messages related to the social structure adaptation, this derivative protocol has some messages related to the norm adaptation (α^N). Briefly, assistants use such messages to exchange information, agree on actual norm updates, and inform participants about new norms.

In particular, each assistant perceives local information about system status and shares a summary ($sump_i$) of this information with other assistants by sending a summary message (`summary <sump>`). Then, they use this information to reason about new desired norm values using their individual norm adaptation functions (α_i^N, see §5.5.3).

Afterwards, assistants agree on actual norms by following a voting scheme ($\beta_{\boldsymbol{\alpha}^N}$). Specifically, assistants propose their desired new max_{FR} value with a vote message (`vote maxFR <value>`) and the max_{BW} value with another vote

message (`vote maxBW <value>`). Then, each assistant computes the voting result —notice this computation is replicated as mentioned in §6.6.3. Finally, when new norms are agreed among assistants, each assistant informs all the participants in its cluster with a message about the actual norm definition (`norm <norm_id> <definition>`).

In addition to inform participants when norms are updated, assistants also notify them when they join the system. For example, in figure 6.5, assistant a_1 sends current norms (`norm <norm_id> <definition>`) to its participants once they join the system. In particular, it does it at $t_{10}..t_{11}$, i.e. before sending them the contacts.

Chapter 7

Adaptation Mechanisms

This chapter elaborates the 2-LAMA adaptation mechanisms in the P2P sharing network case study. In 2-LAMA, the organisational adaptation is an assistance service provided by meta-level agents. In particular, previous chapter refers to the adaptation of two organisational components in the P2P scenario: the net of relationships as part of the social structure and the norms as part of the social conventions. Current chapter details how this adaptation is performed, including two alternatives for the norm adaptation. One alternative is based on a heuristic coded at design-time, whereas the other one is based on a machine learning technique that evolves at run-time. Specifically, the machine learning technique approach is a tailored Case-Based Reasoning technique.

7.1 Introduction

In the context of Coordination Support, we regard the organisational adaptation as an Assistance Layer service as described in chapter 4. From this perspective, chapter 5 proposes the 2-LAMA as an abstract architecture to provide such service. In order to illustrate this approach, chapter 6 presents a specification of the 2-LAMA model in a P2P sharing network scenario. Nonetheless, that chapter just introduces the adaptation mechanisms, whereas current chapter is entirely devoted to detail the adaptation of some organisational components in this scenario.

In particular, this chapter deals with the adaptation of part of the social structure and social conventions. Regarding the social structure, our P2P example adapts the relationships among agents —which are an instantiation of the social structure. Since these relationships pertain to the execution state of the relationships organisational component ($Rels$, eq. 5.6), we refer to this adaptation by using the social relationships adaptation function identifier (α^{Rels}, eq. 5.68). In particular, this adaptation function is based on minimising the network latencies among agents as introduced in §6.6.3. In current implementation, it is performed after the initial protocol phase and each time a computer obtains

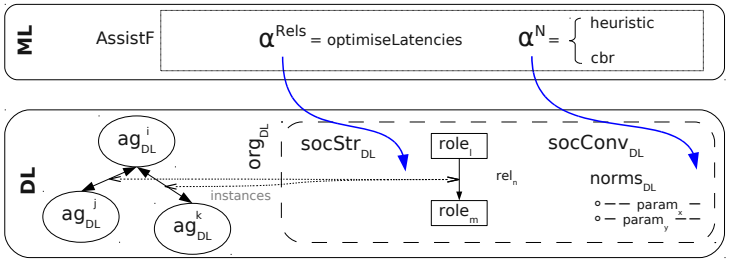


Figure 7.1: Implemented adaptation functions within 2-LAMA context.

the datum —see §6.7.2 and §6.7.4.

On the other hand, regarding the social conventions, the norm adaptation function (α^N, eq. 5.71) adapts some numerical parameters of the norms. In particular, it adapts the maximum allowed values of bandwidth consumption (max_{BW}) and destination computers (max_{FR}) —see §6.5.3. Notice that updating these parameters may be also equivalent to remove those norms, when these limits are large enough to avoid constraining agents —e.g. to let participants use all its nominal bandwidth ($max_{BW} = 100\%$).

Notice that previous chapters formalise a general framework to adapt all organisational components, whereas this chapter provides a detailed description about two of them as a proof of concept. In despite of the potential richness of these two adaptations, the scope of this chapter is to present some versions that illustrate our general model's basic concepts. Accordingly, in order to show that the general model is able to deal with different implementations of these functions, we present two alternatives to perform the norm adaptation as depicted in figure 7.1. As introduced in §6.6.3, one alternative is based on a heuristic (*heuristic*) coded by system designer, while the other one uses machine learning (*cbr*). The former requires previous expert knowledge and provides a fixed estimation. Whereas the latter estimates this relationship automatically and is able to evolve it along time. Specifically, meta-level agents use a tailored Case-Based Reasoning (CBR) [Riesbeck and Schank, 1989] to learn how to adapt the norms. For illustration purposes, figure 7.1 keeps a similar layout and level of generalisation than figure 5.3 —notice that $\alpha^{Rels} \in \boldsymbol{\alpha}^{Rels}$ and $\alpha^N \in \boldsymbol{\alpha}^N$.

Next sections provide further details about each adaptation approach. These approaches are implemented in the P2P simulator described in next chapter 8 and subsequently evaluated in chapter 9.

7.2 Social relationships adaptation

Meta-level agents perform the social structure adaptation function ($\boldsymbol{\alpha}^{Rels}$, eq. 5.68) by applying the corresponding partial adaptation function (α_i^{Rels}, see §5.5.3). More precisely, the described specification ($\alpha^{Rels} \in \boldsymbol{\alpha}^{Rels}$) adapts the actual net of relationships that fulfils the social structure specification, instead

of the sorts of relationships. In other words, in the P2P scenario, the meta-level provides contacts to domain-level agents (i.e. $contact(\mathtt{x},\mathtt{y})$), and these contacts are instances of the sort of specified relationship (i.e. $Contact(a_i, a_j) \in rels$) —see eq. 6.27.

In particular, each assistant is in charge of updating the current net of relationships in its cluster. Specifically, an assistant can request any agent in its cluster to contact other participants in the same cluster or in another one. However, an assistant cannot provide contacts to agents in other clusters. It can only provide complete information to other assistants, who may send contacts to their own participants —see §6.7.4. The resulting domain-level's net of relationships, is the union of all intra-cluster and inter-cluster relationships. That is to say, the agreement function ($\beta_{\alpha^{Rels}} \in \beta_{\alpha^{Rels}}$) is the union of partial relationships adaptation ($\alpha_i^{Rels} \in \alpha_i^{Rels}$) results —see eq. 6.55.

In current implementation, assistants apply their adaptation function during the protocol's social structure phase and every time a domain-level agent is complete —see §6.7.2 and §6.7.4. This way, the net of relationships is created and updated depending on system's evolution.

7.2.1 Information required

In order to perform the social relationships adaptation, an assistant takes into account information about communication latencies and datum possession as described next.

7.2.1.1 Connectivity

On the one hand, an assistant mainly obtains the information about latencies during the social structure protocol phase —see §6.7.2. During this phase, an assistant asks the new agents in its cluster to measure the latencies towards the rest of participants in the same cluster (message `get_lat <peers>`). The received latency measures (message `lat <peer> <measure>`) contribute to the latency environment observable property ($NetLat^C \subset EnvP^C$) and the connectivity metric ($Connect^C \subset EnvP^C$). The former ($NetLat^C$) is used to compute the lower latency paths in order to suggest contacts. Whereas the latter ($Connect^C$), is used to choose the participants that are notified about new complete agents in other clusters —see max_{HAS} in §6.6.2.

In particular, when an assistant receives a new latency measure lat among agent p_x and p_y, it updates $netLat_{p_x,p_y} \in NetLat$ and $netLat_{p_y,p_x} \in NetLat$. In both cases, it computes the average among previous latency average ($netLat_{i,j}$) and current measure (lat) as shown in equation 7.1 —i.e. first $i = p_x, j = p_y$ and afterwards $i = p_y, j = p_x$. This way, the most recent measure is more relevant than the previous ones but it stills keeps long-term information. The resulting latency averages ($\{netLat_{1,1}, \cdots, netLat_{m,m}\}$) conform a weighted graph that is used to compute the lower latency paths as described later in next subsection.

$$netLat_{i,j} = \frac{netLat_{i,j} + lat}{2} \tag{7.1}$$

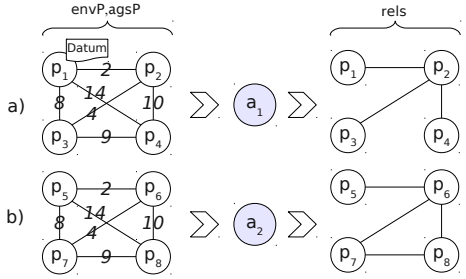


Figure 7.2: Social structure adaptation examples.

In addition to computing $netLat_{p_x,p_y}, netLat_{p_x,p_y}$ when receiving a new latency measure, the assistant also computes the connectivity metrics of the corresponding agents ($connect_{p_x}, connect_{p_y} \in Connect$) as described in equation 7.2 —i.e. first $i = p_x$ and afterwards $i = p_y$. Since, this metric summarises all latency measures from and towards a certain agent, it is computed as an average where all the samples have the same weight. This way, a particular latency to a certain participant cannot bias the result only because it is the last measured.

$$connect_i = \frac{(connect_i \cdot \#samples) + lat}{\#samples + 1} \tag{7.2}$$

7.2.1.2 Datum possession

In addition to connectivity information, an assistant receives information about the datum possession (i.e. $Has, Compl \subset AgsP$) from its cluster but also from other clusters. The local information is obtained at the initial, data sharing, and notification phases —see §6.7.1, §6.7.3 and §6.7.4. In particular, when an assistant receives a join message (`join <hasDatum>`) from participant p_x, it can update has_{p_x} and $compl_{p_x}$. Also, while an agent p_x is receiving the datum, it sends the completeness messages (`completeness <percentage>`) that are used to update $compl_{p_x}$ —see §6.7.4. Finally, when an agent p_x receives the whole datum, the complete message (`complete`) is used to update has_{p_x}.

In addition to local information, the remote one is obtained at the notification phase —see §6.7.4. Specifically, it is obtained when another assistant notifies that a new agent p_y is complete (message `complete_peer <peer>`). In such case, an assistant can update has_{p_y}. It is worth to mention that an assistant has entire information about its cluster (i.e. all $has_i, compl_i$), but only partial information about other clusters (i.e. it only has has_j but not $compl_j$).

7.2.2 Process

The social relationships adaptation process is performed mainly during the *social structure protocol phase* —see §6.7.2. In that period, an assistant faces the two different situations exemplified in figure 7.2. Each situation is shown at left

side of each row (a,b), whereas their resulting net of relationships is shown at the right side. The situations are depicted as weighted graphs that show measured latencies ($netLat_{i,j} \subset envP$) and datum possession ($has_i \subset agsP$). And the resulting net of relationships are depicted as connectivity graphs among participants (instances of $contact(i,j) \in rels$). Both of them are explained as follows.

If some participants within a cluster have the datum (e.g. p_1 in fig. 7.2.a left), the corresponding assistant computes the shortest paths from each agent having the datum to the rest of participants in the cluster. This computation is performed using Dijkstra's algorithm [Dijkstra, 1959] over arc latencies. Next, it re-organises its cluster by telling each participant to contact with its predecessor in its shortest path to a data source (e.g. in figure 7.2.a right, p_3 contacts p_2 because the shortest path is $p_1 \rightarrow p_2 \rightarrow p_3$).

Otherwise, if no participant within a cluster has the datum (e.g. figure 7.2.b left), the corresponding assistant organises its cluster to be prepared for data entering through any agent. Accordingly, it assumes that any participant can become a data source and computes all possible shortest paths —using Dijkstra's algorithm too. Next, it provides each agent with its predecessors in all its corresponding shortest paths (e.g. figure 7.2.b right). This way, all participants are in contact with those neighbours that could rapidly provide the datum when entering through any node in the cluster.

In both cases, the reorganisation is done by sending contact messages (`contact <peers>`) during the social structure protocol phase. In addition, during the *notification phase*, every time a remote participant is complete, the partial adaptation function is invoked again —see §6.7.4. Specifically, when an assistant receives a complete message (`complete_peer <peer>`) the partial function (α_i^{Rels}) determines a new net of relationships in which some local agents can contact the remote one. Then, the assistant sends possession messages (`has_datum <peer>`) to those participants to request that they contact it. Notice that the number of selected agents depends on norm $normHAS$ which limits the amount of these sort of messages —see §6.6.2.

7.3 Norm adaptation

The norm adaptation in the P2P scenario follows the distributed process outlined in 2-LAMA model —in §5.5.3. Briefly, it consists in the four steps depicted in figure 7.3. First, each assistant (a_i) collects some *local information* ($agsP_i^C$, $envP_i^C$) from the participants in its cluster (p_j). This information is related to network consumption and datum possession. Next, they build a summary of this information ($sumP_i$) and exchange it among them. The received summaries constitute their *remote information*. Then, they use all these information to make a *local decision* using their partial norm adaptation functions ($\alpha_i^N \in \boldsymbol{\alpha}_i^N$). Later, their decisions are expressed as desired norm parameter values ($vote_i$) that they exchange according to the agreement function ($\beta_{\boldsymbol{\alpha}^N} \in \beta_{\boldsymbol{\alpha}^N}$). As a result, each assistant computes the average of all parameter values and communicates

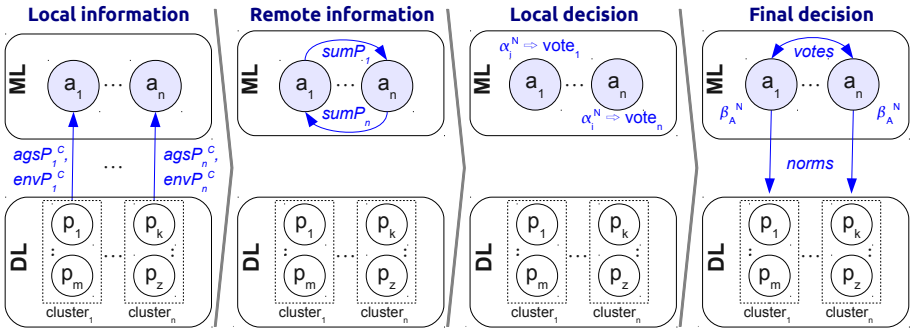


Figure 7.3: Norm adaptation steps.

the actual values to its cluster of participants —see §6.6.3.

This process adapts the maximum allowed values of the two domain-level norms —see §6.5.3. One value is the maximum bandwidth consumption (max_{BW}) which avoids that participants use the network as an infinite resource. And the other value is the maximum number of destination computers (max_{FR}) that regulates the amount of simultaneous data transmissions to avoid network channel saturation. Notice that if assistants set the maximum value of a norm to its upper limit, it is equivalent to remove the corresponding norm.

As discussed in §5.5.4, the frequency of an adaptation process depends on different costs. In the case study, we empirically obtained a time interval ($adapt_{interv}$) after performing different tests in the P2P simulator. This interval is mainly related to the time required for system's stabilisation after a norm update (c_{trans^N}) and the time of collecting the local information (c_{info^N}).

Next subsections detail each step. However, due to the extension and relevance of the two alternatives employed to make the local decision, they are fully described in subsequent sections.

7.3.1 Information required

As detailed in the general model, an assistant takes its decisions based on local information about its cluster and remote information received from other assistants —see §5.5.3.1. The underlying rationale of its decisions is to align the amount of served data with the amount of received data. Thus, the information required consists of measures related to complete and incomplete participants.

7.3.1.1 Local information

On the one hand, each assistant perceives *local information* ($agsP_i^C$, $envP_i^C$) directly from the agents in its cluster. Specifically, in the norm adaptation process, the relevant local information corresponds to the following observable properties (see §6.5.1):

- the datum possession and activity from the agent observable properties ($Has, Compl, Act \subset AgP \subset AgsP^C$)
- and the network consumption from the environment observable properties ($NetBW \subset EnvP^C$).

7.3.1.2 Remote information

On the other hand, each assistant receives *remote information* from other meta-level agents. This remote information is generated by other assistants by applying the summary function ($\sigma_i \in \Sigma$, eq. 5.81) to the local information as formalised in equations 7.4-7.13. The summary ($sumP_i \in SumP$, eq. 7.3) includes the following data derived from local information:

- Regarding the bandwidths:
  - $SeedBW$: the seeds bandwidth is the sum of nominal bandwidth of participants that have the datum —see eq. 7.5, where $(i, j) \in S$ stands for participant j-th of cluster i-th, and $has_{i,j}$ for its datum possession, see eq. 6.12.
  - $LeechBW$: the leeches bandwidth is the sum of nominal bandwidth of participants that lack the datum —see eq. 7.6, where $act_{i,j}$ stands for the activity of participant j-th of cluster i-th, see also eq. 6.14.
  - $SrvBW$: the serving bandwidth is the sum of nominal bandwidths of participants that are serving the datum —see eq. 7.7.
  - $RcvBW$: the receiving bandwidth is the sum of nominal bandwidths of participants that are receiving data —see eq. 7.8.
  - $RcvEffBW$: the effective receiving bandwidth is the sum of actual bandwidths at which participants are receiving data —see eq. 7.9. Notice that the effective bandwidth may be smaller than the nominal one when only a few data are served or if there is network saturation that delays message transport.
  - $AllBW$: the total bandwidth is the sum of all nominal bandwidths —see eq. 7.10.
- Regarding the number of participants:
  - $Waiting$: the waiting agents is the amount of participants that do not have the datum and are neither receiving it —see eq. 7.11.
  - $Compl$: the cluster completeness is the average of completeness of the participants in a cluster —see eq. 7.12, where $compl_{i,j}$ stands for the completeness of participant j-th of cluster i-th, see also eq. 6.13.
  - $NPeers$: the total number of peers is the amount of participants in a cluster —see eq. 7.13.

The total amounts ($AllBW$, $NPeers$) are only used by the learning approach in order to normalise the other metrics.

$$\begin{aligned} SumP = \quad & SeedBW \times LeechBW \times SrvBW \times RcvBW \times \\ & RcvEffBW \times AllBW \times Waiting \times Compl \times NPeers \end{aligned} \tag{7.3}$$

$$\begin{aligned} \sigma(agsP_i^C, envP_i^C) = \quad & sump_i = \\ = \quad & (seedBW_i, leechBW_i, srvBW_i, rcvBW_i, \\ & rcvEffBW_i, allBW_i, waiting_i, compl_i, nPeers_i) \end{aligned} \tag{7.4}$$

$$seedBW_i = \textstyle\sum_{i,j \in S} (nomBW_{i,j}) \ , \ S = \{(i,j) : has_{i,j} = \texttt{yes}\} \tag{7.5}$$

$$leechBW_i = \textstyle\sum_{i,j \in S} (nomBW_{i,j}) \ , \ S = \{(i,j) : has_{i,j} = \texttt{no}\} \tag{7.6}$$

$$srvBW_i = \textstyle\sum_{i,j \in S} (nomBW_{i,j}) \ , \ S = \{(i,j) : act_{i,j} = \texttt{serving}\} \tag{7.7}$$

$$rcvBW_i = \textstyle\sum_{i,j \in R} (nomBW_{i,j}) \ , \ R = \{(i,j) : act_{i,j} = \texttt{receiving}\} \tag{7.8}$$

$$rcvEffBW_i = \textstyle\sum_{i,j \in R} (effDnBW_{i,j}) \ , \ R = \{(i,j) : act_{i,j} = \texttt{receiving}\} \tag{7.9}$$

$$allBW_i = \textstyle\sum_{j=1}^{m_i} (nomBW_{i,j}) \tag{7.10}$$

$$waiting_i = |\{p_{i,j} : p_{i,j} \in ags_i \wedge has_{i,j} = \texttt{no} \wedge act_{i,j} = \texttt{none}\}| \tag{7.11}$$

$$compl_i = \tfrac{\sum compl_{i,j}}{|ags_i|} \tag{7.12}$$

$$nPeers_i = |ags_i| \tag{7.13}$$

7.3.1.3 Knowledge information

Once an assistant receives the remote information, it aggregates such information to the local one in order to obtain the *knowledge information* —see §5.5.3.1. In particular, the knowledge information ($KnowP$) has the local information[1] ($agsP_i^C$,$envP_i^C$) and one component related to each summary component –in eq. 7.3– as shown in equation 7.14 —each component has the identifier of the corresponding summary component plus a 'K' prefix.

$$\begin{aligned} KnowP = \quad & AgsP^C \times EnvP^C \times KseedBW \times KleechBW \times \\ & KsrvBW \times KrcvBW \times KrcvEffBW \times KallBW \times \\ & Kwaiting \times Kcompl \times KnPeers \end{aligned} \tag{7.14}$$

This knowledge is obtained by applying the aggregation function ($\lambda \in \Lambda$, eq. 5.83) specified in equations 7.15-7.17. This function performs a weighted addition of all received remote information ($\{sumP_j \,|\, j = 1..n \wedge j \neq i\}$) and the

[1] Although the norm adaptation just uses the summary of local information, the social structure uses directly the local information —see §7.2.1. Accordingly, the knowledge information also includes the local information. Because, according to 2-LAMA model, all adaptation functions retrieve their information from the knowledge information —see eq. 5.5.3.2.

summary of local information ($sumP_i = \sigma(agsP_i^C, envP_i^C)$) —where i stands for the i-th assistant. Essentially, each of these summary-related components is the sum of a local summary component plus remote ones using different weights. For instance, equation 7.16 illustrates how the knowledge serving bandwidth of a given assistant ($ksrvBW_i$) is computed. It is the local serving bandwidth ($srvBW_i$) multiplied by a *local weight* (w_L) plus the sum of every remote serving bandwidth ($\{srvBW_j \,|\, j \neq i\}$) multiplied by *remote weights* ($w_{R,j}$).

$$\begin{aligned}\lambda(agsP_i^C, envP_i^C, envP^{SnC}, \{sumP_x : x = 1..n \wedge x \neq i\}) = \\ = knowP_i = \big(agsP_i^C, envP_i^C, kseedBW_i,\ kleechBW_i, ksrvBW_i, \\ krcvBW_i,\ krcvEffBW_i, kallBW_i, kwaiting_i, \\ kcompl_i, knPeers_i\big)\end{aligned} \tag{7.15}$$

$$ksrvBW_i = w_L \cdot srvBW_i + \textstyle\sum_{j=1}^{n} \{w_{R,j} \cdot srvBW_j : j \neq i\} \tag{7.16}$$

$$\Big(\big(w_L + \textstyle\sum_{j=1}^{n} \{w_{R,j} | \, j \neq i\}\big) = 1\Big) \wedge (\nexists w_{R,j} \,|\, w_{R,j} > w_L) \tag{7.17}$$

The sum of all these weights is one as shown in equation 7.17, so the result has the same range as the original components. Moreover, this equation shows that we assume that remote information cannot be more relevant than local one. For example, if local weight has its maximum value ($w_L = 1$), then each assistant takes into account only its cluster status. In contrast, if this weight is the minimum ($\forall_j w_{R,j} = w_L$), then each assistant gives the same importance to local information as to the remote information —this is the case in current tests. The mid-point is an imbalance importance among local and remote information that leads an assistant to take its decisions giving more importance to its local cluster, but taking into account the rest of the system.

7.3.2 Local decision

In order to make their local decision, each assistant uses its partial norm adaptation function ($\alpha_i^N \in \mathcal{A}_i^N$, eq. 5.84). Such function estimates a relationship among current norms and system performance, with the aim to suggest norm updates.

In order to show that the model is able to deal with different implementations of these functions, we present two alternative functions —see §6.6.3. The first alternative ($\alpha_i^N = heuristic$) is based on a heuristic coded by the system designer. Basically, such heuristic tries to align the amount of serving capacity with the receiving capacity. Accordingly, it requires previous expert knowledge and provides a criteria that is fixed at design-time —i.e. it always uses the same coded algorithm to adapt norms.

In contrast, the second alternative ($\alpha_i^N = cbr$) is based on a machine learning technique. In particular, it is a tailored Case-Based Reasoning (CBR) [Riesbeck and Schank, 1989], so it uses a base of previous cases to decide on new situations. The tailored machine learning technique lets an assistant estimate the relationship among norms and performance automatically at run-time and lets the assistant evolve it along time.

Due to the extension and relevance of the two alternatives, they deserve an entire section devoted to provide all their details. Hence, they are fully described in sections 7.4 and 7.5. In addition to the presented approaches, there could exist other alternatives. Even more, each assistant could make its local decision using a different approach. The only constraint is that they have the domain and co-domain specified in eq. 5.84. In such a case, it is easy to add them to the developed simulator described in next chapter. In other words, as all norm adaptation approaches take the same information and provide the same sort of result, the resulting performance of the system can be compared.

7.3.3 Final decision

After each assistant computes its desired update for norm parameters all of them agree on their actual modification. Moreover, they actually update the norm and notify it to domain-level agents. As a result, agent interactions are influenced by these new norms. These agreement and norm adoption processes are detailed as follows.

7.3.3.1 Agreement

The agreement function ($\beta_{\alpha^N} \in \beta_{\alpha^N}$, eq. 5.85) specified in the P2P scenario is a voting approach —see 6.6.3. Accordingly, an assistant expresses its desired update in the form of one vote for each norm. That is, one vote for the value of the max_{FR} parameter of $normFR$ and another one for the max_{BW} parameter of $normBW$. Both values may include a special value that represents a *blank-ballot paper* (`BLNK`). This special value means that no update is suggested in order to let the other assistants push for their own interests. In such a case, the other assistants will better choose required norm adaptations.

Next, each assistant sends its votes to other assistants using the vote messages —`vote maxFR <value>` and `vote maxBW <value>`, see §6.7.5. After, when it receives the votes of all other assistants, it computes the most frequent vote for each parameter discarding blank ballot-papers —i.e. there is a distributed implementation that replicates the ballot recount. Finally, if norms are actually updated, then the assistant sends them to its domain-level agents by means of the norm definition messages (`norm <norm_id> <definition>`).

7.3.3.2 Norm adoption

Once a domain-level agent receives new norms, it tries to fulfil them —notice that participants do not violate norms in current implementation. Thus, when an agent receives a new max_{BW} smaller than the one it is using, it decreases its sending ratio to fulfil this norm. Besides, when it receives a new max_{FR}, it also tries to fulfil it. This means that if an agent is serving fewer participants than max_{FR}, it will send and unblocking message (`unchoke`) to those participants it had blocked previously —see §6.7.3. In contrast, if it was serving to more friends than max_{FR}, it will cancel some of those data transmissions and send

a blocking message (`choke`). Nevertheless, in our current implementation, an agent does not need to cancel a data transmission if it has already sent more than 75% of the datum. This behaviour avoids cancelling data transmissions that will finish really soon in order to minimise c_{adopt^N} and c_{trans^N} —see §5.5.4.

7.4 Heuristic approach to norm adaptation

The first alternative to perform the norm adaptation is a heuristic coded by the system designer based on previous expert knowledge. In the P2P case study, this knowledge was gained by observing several sharing processes with the same initial conditions but different norm parameter values. As a result, the empirically found heuristic tries to align the amount of serving capacity with the receiving capacity. This heuristic is coded once at design-time and it is used along all sharing processes. In other words, this alternative is fixed a-priori and it does not evolve, whereas the machine learning approach described in next section updates its criteria depending on its experience.

According to our 2-LAMA model, the heuristic partial norm adaptation function ($heuristic \in \alpha_i^N$) establishes a relationship among knowledge information ($KnowP$), goals ($Goals$), current norms (first $Norms$ label), and new norms (last $Norms$ label) as formalised in equation 7.18.

$$heuristic : KnowP \times Goals \times Norms \rightarrow Norms \tag{7.18}$$

Specifically, the implemented heuristic function ($heuristic'$) takes into account only some components of the knowledge information and norms, and returns the sort of changes to apply to norms ($Changes$) as formalised in equations 7.19-7.20.

$$\begin{aligned} heuristic' : \quad & KsrvBW \times KrcvBW \times KrcvEffBW \times \\ & Kwaiting \times Norms \rightarrow Changes \end{aligned} \tag{7.19}$$

$$\begin{aligned} Changes = \quad & ChangesFR \times ChangesBW \\ & ChangesFR = \{\texttt{DECR}, \texttt{SAME}, \texttt{INCR}, \texttt{BLNK}\} \\ & ChangesBW = \{\texttt{DECR}, \texttt{SAME}, \texttt{MAX}\} \end{aligned} \tag{7.20}$$

In particular, the heuristic may suggest the following $normFR$ changes ($ChangesFR$): to decrease max_{FR} by 1 (`DECR`), to leave the same value (`SAME`), to increase it by 1 (`INCR`), or to change it to the special blank-ballot value (`BLNK`, see 7.3.3.1). Besides, it may suggest the following $normBW$ changes ($ChangesBW$): to divide max_{BW} by 2 (`DECR`), to leave the same value (`SAME`), or to set it to 100% (`MAX`).

As illustrated in algorithm 7.1, the heuristic function ($heuristic$, eq. 7.1) receives the expected parameters –see line 1– and calls the implemented heuristic ($heuristic'$, eq. 7.19) to obtain the changes to apply to norms —see line 3. Next, it applies those changes depending on the semantic described in previous paragraph —see lines 8-9. Finally, it returns new norms —line 11.

Algorithm 7.1 Heuristic approach of partial norm adaptation function.

```
01 def heuristic( knowP, goals, norms ):
02
03   changes = heuristic'( knowP.ksrvBW,    knowP.krcvBW,
04                         knowP.krcvEffBW, knowP.kwaiting,
05                         norms.maxFR,     norms.maxBW )
06
07   newNorms = new Norms()
08   newNorms.maxFR = apply( changes.FR, norms.maxFR )
09   newNorms.maxBW = apply( changes.BW, norms.maxBW )
10
11   return newNorms
```

Besides, the implemented heuristic function (*heuristic'*) is schematised in algorithm 7.2. This algorithm receives the mentioned knowledge components plus current norms expressed by their parameter values —see line 1. In addition, it uses information derived from combining both knowledge and current norms —line 3. This additional information is the *expected receiving bandwidth* ($RcvExpBW$), that re-scales receiving nominal bandwidth ($RcvBW$) according to current bandwidth limit (max_{BW}) as expressed in equation 7.21. This new information reflects that actual receiving bandwidth may be lower when there is a bandwidth limit applied to serving agents —since less data is being injected towards receiving participants.

$$rcvExpBW_i = rcvBW_i \cdot \frac{max_{BW}}{100} \tag{7.21}$$

Also at the beginning, the algorithm starts initialising some constants that are used as thresholds in comparisons and the changes that will be returned —line 3. Next, the main decision regarding the choice of a $normFR$ is related to comparing the available bandwidth used to serve (`srvBW`) with the available bandwidth used to receive (`rcvBW`). If there is a lack of serving bandwidth, the suggestion is to decrease the number of friends max_{FR} —line 7. This way, server agents will be simultaneously serving data to fewer participants, and these transmissions will finish sooner. Afterwards, once these other participants have the datum, there will be more data sources in the system and it will take less time to finish the datum distribution. On the other hand, if there is an excess of serving bandwidth and there are still participants waiting for data, then the assistant can increase the number of friends in order to serve more agents —lines 9-10. There is another situation in which there is also an excess of serving bandwidth but there are no participants waiting for data —lines 12-13. This does not necessarily mean all agents have the datum, but at least the ones lacking it are receiving it from some source. In this case, the assistant uses a blank-ballot paper to let other assistants push for their own interests[2].

[2]Notice, though, that the weighting method applied to measures (see §7.3.1.3) may bring an assistant to this case when no participants in its cluster are waiting for data, but there are

Algorithm 7.2 Implemented heuristic.

```
01 def heuristic'( srvBW, rcvBW, rcvEffBW, waiting, maxFR, maxBW ):
02
03   rcvExpBW = rcvBW * (maxBW / 100)
04   τ = 0.1 ; ϵ = 0.2 ; changes = new Changes()
05
06   // Adapt maxFR ------------
07   case  ( srvBW < (1-τ)*rcvBW )  : changes.FR = DECR
08
09   case  ( srvBW > (1+τ)*rcvBW
10                &&   waiting > ϵ ): changes.FR = INCR
11
12   case  ( srvBW > (1+τ)*rcvBW
13                &&   waiting < ϵ ): changes.FR = BLNK
14
15   other /* srvBW ≈  rcvBW */    : changes.FR = SAME
16
17
18   if( rcvEffBW <(1-τ)*rcvExpBW ) : changes.FR = DECR
19
20   // Adapt maxBW ------------
21   case (changes.FR = DECR ∧ maxFR = 1   : changes.BW= DECR
22   case (changes.FR = INCR ∧ maxBW < 100): changes.BW= MAX
23   other                                 : changes.BW= SAME
24
25   return changes
```

Finally, if none of the previous cases holds, it means that the serving bandwidth is similar to the receiving one. Then, the assistant opts for keeping the same norm —line 15. That is to say, if there is no excess of serving bandwidth, the assistant opts for the same norm instead of just leaving the decision to the rest of the assistants.

In spite of above cases, if there is network saturation in the intermediate channels, it is always better to decrease the number of friends. This will reduce the number of data transmissions. Hence, it will cut back network traffic and hopefully network saturation. In order to estimate whether there is network saturation, the assistant checks whether the effective receiving bandwidth (`rcvEffBW`) is smaller than the expected one (`rcvExpBW`). If so, this suggests that data packets are delayed by the intermediate network because it is saturated. Consequently, as a solution to saturation, the assistant opts for decreasing max_{FR} —line 18.

As for the $normBW$, max_{BW} is only decreased if it is not possible to reduce the network usage further by decreasing the number of friends —since max_{FR}

still waiting agents in other clusters. In such a case, if there is enough serving bandwidth, it is better to let other assistants choose by themselves the norm parameter values.

is already 1. In such a case, the assistant opts for decreasing max_{BW} —line 21. This way, server agents will use less bandwidth, which can help to diminish the network saturation. In contrast, if the bandwidth was previously limited but there is no network saturation –since the assistant chose to increase max_{FR}–, then the bandwidth limit can be reset to 100% —line 22. For the remaining cases, max_{BW} keeps its value —line 23.

7.5 Machine Learning approach to norm adaptation

The second alternative to perform the norm adaptation is a machine learning technique. As opposed to previous static heuristic, this machine learning approach evolves its behaviour depending on its experience. In particular, it is a tailored Case-Based Reasoning (CBR) technique. Next subsection highlights some characteristics of the P2P case study considering the machine learning perspective, and specify how CBR technique is tailored to deal with such a kind of scenario.

7.5.1 Characterisation

According to our 2-LAMA model, the partial norm adaptation function (α_i^N, eq. 7.22) establishes a relationship among knowledge information ($KnowP$), goals ($Goals$), current norms (first $Norms$ label), and new norms (last $Norms$ label).

$$\begin{array}{lcccc} \alpha_i^N & : & KnowP \times Goals \times Norms & \rightarrow & Norms \\ relation & : & State & \rightarrow & Actions \end{array} \tag{7.22}$$

In other words, it is a relation among system state (estimated by $KnowP$ under $Goals$ perspective) and the convenient actions (influenced by new $Norms$). This mapping is highly complex, as stated by the cooperative MAS learning taxonomy in [Panait and Luke, 2005], since agent interactions may bring unexpected joint behaviour. On the one hand, this categorisation defines as *team learning* a centralised approach to discover a set of behaviours for a set of agents. On the other hand, it classifies as *concurrent learning* those approaches where there are multiple learners. They require that the search space can be split in disjoint parts that require disjoint actions —i.e. to decompose the problem and the solution. However, our case joins both approaches, because we look for a distributed learning (i.e. multiple learners) about an organisational level instead of a local one (i.e. the search space cannot be split).

Furthermore, in the P2P scenario, this relationship is totally undetermined since the most *appropriate norms are unknown* for each adaptation interval. However, it is still possible to obtain feedback about system evolution once a norm adaptation has been applied. Besides, the time to spread the datum

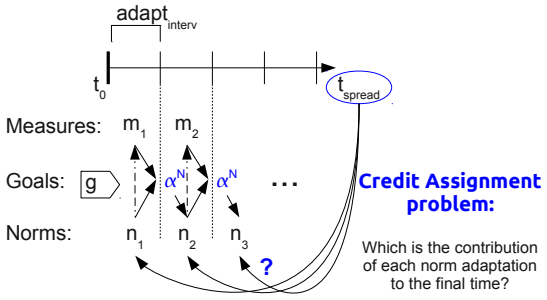


Figure 7.4: Credit assignment problem.

among all agents (t_{spread}) depends, among others, on the several norm adaptations performed along the execution. This poses a *credit assignment problem* [Jones and Goel, 2004] since it is difficult to identify which is the positive or negative influence of each norm adaptation on t_{spread}. Figure 7.4 illustrates this problem. From the beginning of the sharing process (t_0), norms are adapted at certain intervals ($adapt_{interv}$) depending on system state (m_i evaluated by goals g). Once the sharing process is finished, there is a credit assignment problem to determine which norms have influenced positively or negatively on the final time (at t_{spread}). In addition, to these norm issues, the P2P scenario fulfils the *similarity assumption* as, in general, similar norms are useful to close situations.

Lastly, the state space is very large, since it is *multidimensional* and *continuous*. For instance, it has more than one bandwidth measure, and all them are continuous. Similarly, the *action space* is also multidimensional and has one continuous dimension —the max_{BW} norm's parameter.

In order to face all these characteristics, we have tailored a Case-Based Reasoning (CBR). The CBR method is defined as the process of solving new problems by retrieving the solutions to the most similar problems from a knowledge-base and adapting them to new problems [Riesbeck and Schank, 1989]. It is able to deal with multidimensional continuous spaces taking profit of the similarity assumption. Furthermore, we extend it to be able to take into account heuristic suggestions and to tackle the lack of supervision by considering its own experience —i.e. previously solved problems.

7.5.2 Case description

In classical CBR, the knowledge is represented as a collection of cases ($CaseBase$) where a case ($Case$) has two components: a problem ($Prob$) and its solution (Sol). As figure 7.5 shows, we add a third component representing the feedback about system evolution: the evaluation ($Eval$) of a solution. Such a case is formalised in equation 7.23.

$$Case = Prob \times Sol \times Eval \tag{7.23}$$

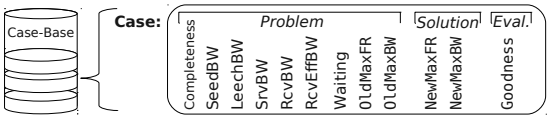


Figure 7.5: CBR's case description.

A *problem* description consists of attributes (*Attrib*) derived from knowledge information ($KnowP$) and current norms ($Norms$). Specifically, there are the nine numeric attributes shown in equation 7.24. Those attributes related to the knowledge information have the same identifier than in $KnowP$ without the 'K' prefix —see §7.3.1.3. Whereas the two attributes related to norms have an 'Old' prefix to distinct them from the new values contained by the solution attributes.

$$Prob = Completeness \times SeedBW \times LeechBW \times SrvBW \times RcvBW \times RcvEffBW \times Waiting \times OldMaxFR \times OldMaxBW \tag{7.24}$$

$$completeness = {}^{k}compl_i/_{100} \tag{7.25}$$

$$seedBW = {}^{k}seedBW_i/_{kallBW_i} \tag{7.26}$$

$$waiting = {}^{k}waiting_i/_{knPeers_i} \tag{7.27}$$

$$oldMaxFR = {}^{max_{FR}}/_{knPeers_i} \tag{7.28}$$

$$oldMaxBW = {}^{max_{BW}}/_{100} \tag{7.29}$$

In order to allow to compare similar cases in despite of network scale, all attributes are normalised to fit in the range [0..1]. In particular, the bandwidth measures are divided by the sum of the nominal bandwidth of all participants —e.g. eq. 7.26. The values related to the the amount of participants are divided by the total number of domain-level agents —e.g. eq. 7.28. And the percentage values are divided by one hundred —e.g. eq. 7.29.

As for the *solution*, it has two numeric attributes as formalised in equations 7.30-7.32. They are normalised to fit in the range [0..1], too. The former ($NewMaxFR$) corresponds to the updated max_{FR} normalised by the number of participants. And the latter ($NewMaxBW$) corresponds to the updated max_{BW} normalised by one hundred. Both attributes may have the blank-ballot paper value (`BLNK`) meaning that the assistant does not require any specific value. An assistant may choose this value if no participants in its cluster are waiting for data and other clusters still contain waiting agents. In such a case, their assistants will better choose required norm adaptations.

$$Sol = NewMaxFR \times NewMaxBW \tag{7.30}$$

$$newMaxFR = \frac{max_{FR}}{knPeers_i} \vee newMaxFR = \texttt{BLNK} \tag{7.31}$$

$$newMaxBW = \frac{max_{BW}}{100} \vee newMaxBW = \texttt{BLNK} \tag{7.32}$$

Last, the *evaluation* has a single attribute (*Goodness*) which provides an evaluation of how effective is the corresponding solution. As next subsections 7.5.3.3-7.5.3.4 detail, this metric takes into account the increment of *Completeness* and t_{spread}. Also, like other case attributes its range is [0..1] —see eq. 7.33-7.34.

$$Eval = Goodness \tag{7.33}$$

$$Goodness = \{n : n \in \mathbb{R} \wedge n \in [0..1]\} \tag{7.34}$$

7.5.3 CBR Cycle

The classical CBR cycle [Aam, 1994] has four phases: retrieve, reuse, revise and retain. We tailored these phases in order to take into account heuristic suggestions and to let CBR consider its own experience when applying a solution to the P2P sharing network. The resulting cycle is depicted in figure 7.6. Briefly, once a new problem is encountered, the first phase *retrieves similar* cases from the case-base —if there are no enough similar problems, the heuristic is called to provide a solution. Next, the second phase *reuses* the retrieved case(s) to provide a solution to the new problem —again, if the solutions of retrieved cases are too divergent, the heuristic is called instead. Then, each assistant participates in the agreement process (β_{α^N}) by voting for the solution returned by its CBR reuse phase. Subsequently, the agreed solution is applied —i.e. the norms are updated. After a certain time interval ($adapt_{interval}$), the third phase *revises* the results of applying the new solution. And finally, the fourth phase *retains* the new problem if it is representative enough.

Hence, the learning partial norm adaptation ($cbr \in \alpha_i^N$, eq. 7.35) is a composition of the functions formalised in equations 7.36-7.40. Among them, the heuristic ($heuristic''$) is equivalent to the function described in previous section ($heuristic'$), but it extracts its information from a problem definition, and it is able to return its results as a CBR case, instead.

$$cbr : KnowP \times Goals \times Norms \rightarrow Norms \tag{7.35}$$

$$retrieve : Prob \times CaseBase \rightarrow (Case)^* \tag{7.36}$$

$$reuse : Prob \times (Case)^* \rightarrow Sol \tag{7.37}$$

$$revise : Solution \times KnowP \rightarrow Eval \tag{7.38}$$

$$retain : Case \times (Case)^* \times CaseBase \rightarrow CaseBase \tag{7.39}$$

$$heuristic'' : Prob \rightarrow Case \tag{7.40}$$

Specifically, algorithm 7.3 illustrates how *cbr* composes the mentioned functions. This algorithm receives the knowledge information, the goals, and current norms —see line 1. Since it implements the cycle described in figure 7.6, it checks if there was already a previous case —line 3. If the cycle has just started, there is

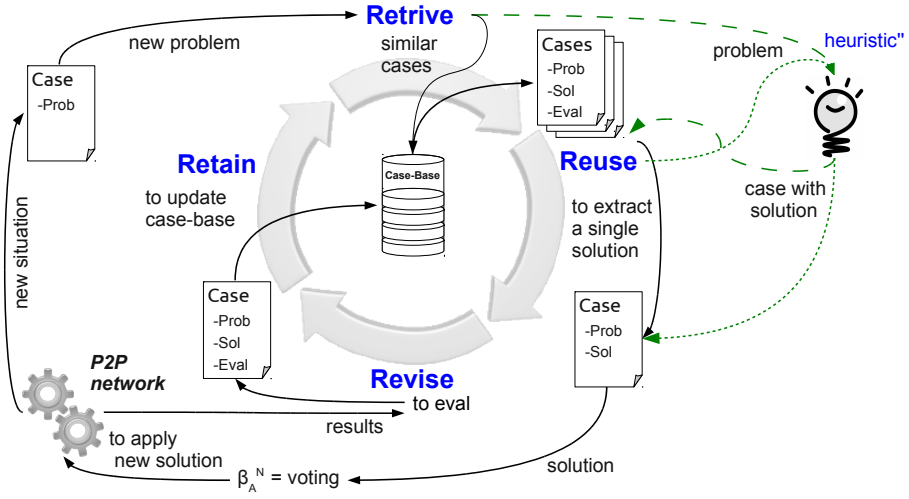


Figure 7.6: Tailored CBR cycle.

still no previous case. In such a case, the algorithm continues by creating a new case from received information —line 7. In fact, this new case only has initialised the problem description. Such description (`newCase.prob`) is sent to the retrieve function to fetch similar cases (`retrCases`) from the case base —line 9. As detailed in §7.5.3.1, if there are no cases in the case-base (`caseBase`), or they are not similar enough to current problem, the retrieve function may use the heuristic. Afterwards, the algorithm sends these cases and the original problem to the reuse function in order to extract a solution —line 11. This solution (`newCase.sol`) is obtained as described in §7.5.3.2. It contains the values of both norm parameters normalised in the range [0..1]. Hence, the algorithm builds a new norm set (`newNorms`) by multiplying the max_{FR} parameter (`newMaxFR`) by the number of participants (`knowP.knPeers`) and the max_{BW} parameter (`newMaxBW`) by one hundred —lines 13-15. Before returning these new norms, the algorithm temporally keeps track of current case (`prevCase`) and retrieved cases (`prevRetrCases`) to be used in *cbr* function invocation —line 17. Next, these new norms are returned –line 19– since they are the result of the partial norm adaptation ($\alpha_i^N \in \alpha_i^N$).

The corresponding assistant uses this result in the voting scheme ($\beta_{\alpha^N} \in \beta_{\alpha^N}$) described in §7.3.3. As illustrated in figure 7.6, norms may be updated and bring the system towards a new situation. After the adaptation interval ($adapt_{interv}$), the *cbr* function is called again. This time, the initial check about previous case is true —line 3. Accordingly, the algorithm sends the previous solution (`prevCase.sol`) and current system status (`knowP`) to the revise function —line 4. Such function returns an evaluation (`prevCase.eval`) of system status as described in §7.5.3.2. Next, the retain function receives all the information about previous phases and updates the case-base as detailed in §7.5.3.4 —line 5. Finally, the cycle continues by the retrieve phase. In addition to the

Algorithm 7.3 CBR approach of partial norm adaptation function.

```
01 def cbr( knowP, goals, norms ):
02
03   if ( prevCase ):
04     prevCase.eval = revise( prevCase.sol, knowP )
05     caseBase      = retain( prevCase, prevRetrCases, caseBase )
06
07   newCase = new Case( knowP, goals, norms )
08
09   retrCases = retrieve( newCase.prob, caseBase )
10
11   newCase.sol = reuse( newCase.prob, retrCases )
12
13   newNorms = new Norms()
14   newNorms.maxFR = newCase.sol.newMaxFR * knowP.knPeers
15   newNorms.maxBW = newCase.sol.newMaxBW * 100
16
17   prevCase = newCase ; prevRetrCases = retrCases
18
19   return newNorms
```

sequence, at the end of the sharing process there is a last call to the revise and retain functions in order to evaluate the last applied solution.

Next subsections provide further details about each phase and how the classical CBR is tailored.

7.5.3.1 Retrieve

Once a new problem is encountered, the first phase *retrieves similar* cases from the case-base —eq. 7.36. It returns the k most similar previous cases that have at least a *minimum similarity* (`MIN_SIM`) to current problem. This phase may return less than k cases if there are not enough similar cases. If no cases at all are retrieved, then the heuristic ($heuristic''$) is used to solve current problem —see eq. 7.40 and §7.4.

Algorithm 7.4 illustrates this process. It starts with an empty list of cases (`retrCases`) —see line 3. Then, it traverses the case base –line 5– computing the similarity (Θ, see eq. 7.41) of each previous case's problem description (`prevCase.prob`) with the new problem (`prob`) —line 6. In case this similarity is greater than the minimum trusted similarity (`MIN_SIM`), the case is collected —lines 7-8. However, if no previous case has the minimum trusted similarity to consider it is representative enough to adapt its solution to the new problem –line 10–, the algorithm executes the heuristic –line 11– to solve current unknown problem. Finally, the cases are returned —line 14.

The implemented similarity function ($\Theta \in [0..1]$) among two problems ($p_x, p_y \in Prob$) is computed like shown in equation 7.41. It is one less the Euclidean distance among attribute values ($a_i^{p_x}, a_i^{p_y} \in Attrib_i$) aggregated in a

Algorithm 7.4 Retrieve algorithm.

```
01 def retrieve( prob, caseBase ):
02
03  retrCases = ∅
04
05  foreach prevCase in caseBase:
06    s = Θ( prevCase.prob, prob )
07    if ( s > MIN_SIM ):
08      retrCases = retrCases ∪ { prevCase }
09
10  if ( retrCases = ∅ ):
11    heuCase   = heuristic''( prob )
12    retrCases = { heuCase }
13
14  return retrCases
```

weighted manner (w_i^{Θ}). Currently, weights are computed using the Proportional Rough Sets (PRS) [Salamó and López-Sánchez, 2011] method.

$$\Theta(p_x, p_y) = 1 - \sqrt{\frac{\sum_{i \in Attribs} \left(w_i^{\Theta} \cdot |a_i^{p_x} - a_i^{p_y}| \right)}{\sum_{i \in Attribs} w_i^{\Theta}}} \tag{7.41}$$

Notice that in the P2P scenario the case-base is initially empty. Hence, the retrieve phase starts by invoking the heuristic. However, once there exist some previous cases, the use of the heuristic is marginal as shown in chapter 9.

7.5.3.2 Reuse

Second phase *reuses* the retrieved case(s) to provide a solution to the new problem —eq. 7.37. Our reuse phase starts by checking if the divergence of retrieved solutions is greater than a *maximum trusted divergence* (MAX_DIV) threshold. This divergence (δ) is computed as the standard deviation of $NewMaxFR$ solution's attribute —since in our experiments $NewMaxBW$ was correlated with it. Note that the divergence of a single case is 0. Exceeding MAX_DIV means that the solutions of retrieved cases are too contradictory to provide a good single solution for current problem. In such a case, the heuristic ($heuristic''$) is invoked to obtain a solution —see eq. 7.40 and §7.4. Once there is a set of slightly divergent retrieved cases, it performs an adaptation of the solution of these cases.

Algorithm 7.5 illustrates this process. It starts by checking if the divergence (δ) of retrieved cases (retrCases) is greater than the maximum trusted divergence (MAX_DIV) —see line 3. In such a case, it considers that previous cases' solutions are too contradictory to provide a good single solution. Hence, the heuristic is used —lines 4-5. Once there is a set of slightly divergent previous cases –remind that a single previous case has no divergence– it adapts their solution to the current problem —line 7. Finally, the solution is returned —line 9.

The adaptation of the solution of retrieved cases (*adapt*) can take into account (i) all retrieved solutions but also (ii) the differences between the retrieved

Algorithm 7.5 Reuse algorithm.

```
01 def reuse( prob, retrCases ):
02
03  if ( δ(retrCases) > MAX_DIV )
04    heuCase = heuristic''( prob )
05    retrCases  = { heuCase }
06
07  sol = adapt( prob, retrCases )
08
09  return sol
```

problems and the current one. Currently, the *adapt* function uses only the former (i) as expressed in eq. 7.42-7.48.

$$adapt(prob, retrCases) = (wAvg(FR, G), wAvg(BW, G)) \tag{7.42}$$

$$FR = \{c.sol.\texttt{max}_{\texttt{FR}} | c \in retrCases\} \tag{7.43}$$

$$BW = \{c.sol.\texttt{max}_{\texttt{BW}} | c \in cases\} \tag{7.44}$$

$$G = \{c.eval.\texttt{goodness} | c \in cases\} \tag{7.45}$$

$$wAvg(V, G) = \begin{cases} \texttt{BLNK} & if\ (gB > gNB) \\ \frac{\sum(v_i \cdot g_i | v_i \neq \texttt{BLNK})}{gNB} & otherwise \end{cases} \tag{7.46}$$

$$gB = \textstyle\sum_{(v_i, g_i) \in (V,G)} (g_i | v_i = \texttt{BLNK}) \tag{7.47}$$

$$gNB = \textstyle\sum_{(v_i, g_i) \in (V,G)} (g_i | v_i \neq \texttt{BLNK}) \tag{7.48}$$

Briefly, the *adapt* function receives the set of retrieved cases (*retrCases*) and returns a solution that is the average of retrieved cases' solutions weighted by their evaluation —notice that it is not using the current problem. In more detail, this function traverses the retrieved cases and collects their $\texttt{max}_{\texttt{FR}}$ (FR, eq. 7.43), their $\texttt{max}_{\texttt{BW}}$ (BW, eq. 7.44) and their evaluation metric (G, eq. 7.45). Then, it computes each solution's dimension by averaging their values depending on their evaluation (i.e. their *Goodness*) using a weighted average function ($wAvg$, eq. 7.46). This function receives a set of values (V, i.e. FR or BW) and their associated weights determined by their goodness (G). Then, if the goodness of blank values (gB, eq. 7.47) is greater than the goodness of the non-blank values (gNB, eq. 7.48), the result is a blank-ballot paper (`BLNK`, see §7.5.2). Otherwise, the result is the weighted average of the non-blank values, by giving more importance to the values with higher goodness (i.e. the sum of non-blank values times their goodness, normalised by the sum of their goodness).

Algorithm 7.6 illustrates the described *adapt* function. Briefly, it initialises the sets that will contain each solution's dimension (`ValuesFR`, `ValuesBW`), the set that will contain the evaluation of applying them in the past (`Goodns`), and the new solution —see line 3. Next, it traverses the retrieved cases –line 5– and fills the three mentioned sets. That is, the `ValuesFR` with corresponding

Algorithm 7.6 Adaptation algorithm.

```
01 def adapt( prob, retrCases ):
02
03  ValuesFR=∅; ValuesBW=∅; Goodns=∅ ; sol = new Solution()
04
05  for c in retrCases:
06    ValuesFR = ValuesFR ∪ {c.sol.maxFR}
07    ValuesBW = ValuesBW ∪ {c.sol.maxBW}
08    Goodns   = Goodns ∪ {c.eval.goodness}
09
10  sol.maxFR = wAvg(ValuesFR, Goodns)
11  sol.maxBW = wAvg(ValuesBW, Goodns)
12
13  return sol
```

$\texttt{max}_{\texttt{FR}}$, the `ValuesBW` with $\texttt{max}_{\texttt{BW}}$ and the `Goodns` with the evaluation metric of the corresponding case —lines 6-8. Then, it computes each solution's dimension by averaging their values depending on their evaluation (i.e. their *Goodness*) using the $wAvg$ function —lines 10-11. Finally, the resulting values conform the returned solution —line 13.

After computing the solution for the current case, each assistant participates in the agreement process (β_{α^N}, eq. 5.85) by sending a vote for each norm to the other assistants. Hence, assistants receive other's votes and compute the most frequent vote for each norm discarding blank ballot-papers (if there is a tie, norms are not updated). Then, each assistant communicates the new norms to its domain-level agents. Such norms are used during a time interval ($adapt_{interval}$) until next adaptation process begins with the subsequent phase.

7.5.3.3 Revise

As mentioned above, the adaptation process starts at the revise phase, which requires to evaluate the applied solution —eq. 7.38. However, current *Goals* definition is related to the total spread time (t_{spread}) which is unknown until the end of the sharing process. Alternatively, our evaluation metric ($g \in [0..1]$) is based on the datum possession percentage (*Completeness*) since it is related to t_{spread}: if the increment of *Completeness* (c_{inc}) along an adaptation interval ($adapt_{interval}$) is large, it means that a significant number of peers increased their percentage of the datum, so a short t_{spread} is expected.

$$g = \frac{(c_{inc} - min_{inc})}{(max_{inc} - min_{inc})} \tag{7.49}$$

In particular, g is the *Completeness* increment (c_{inc}) normalised by the difference between the maximum and minimum values of c_{inc} that could be obtained during this specific adaptation interval —see eq. 7.49. The max_{inc} value refers to the maximum c_{inc} if all seeds serve –and leeches also receive– at their nominal

bandwidth. The min_{inc} value indicates the minimum c_{inc} if all data transmissions are cancelled. Overall, cases with the largest goodness are expected to provide better solutions. Accordingly, this evaluation is used by the reuse phase to assign different weights to retrieved solutions.

7.5.3.4 Retain

Fourth CBR phase retains a new problem if it is representative enough. In particular, once a solution has been revised, the information about this experience is added to the case-base or used to update an existing case —eq. 7.39. In fact, when a solution was generated by the heuristic (either at retrieve or reuse phases), it is stored as a new case which includes its problem description and its evaluation. Likewise, when a solution was created from more than one existing case, it is added as a new case. In contrast, when a solution comes from a single similar case, the evaluation of this previous case is updated following the constant-α Monte Carlo strategy [Sutton and Barto, 1998] shown in equation 7.50. This way, the new goodness stored in the case-base (g_{new}) is the previous one (g_{prev}) updated according to the current experience (g_{cur}) depending on the given learning rate constant (α_g).

$$g_{new} = \alpha_g \cdot g_{cur} + (1 - \alpha_g) \cdot g_{prev} \tag{7.50}$$

In addition to considering the completeness degree, at the end of the whole sharing process, this phase also updates (see eq. 7.51) the goodness of all used cases during the CBR process by considering the final data spread time (t_{spread}). Notice that the goodness (g) of each used case is increased by a given factor (k_g) if this time is smaller than the average sharing time (t_{avg}, which is the average of previous t_{spread}). Otherwise, it is decreased. This strategy aims to include information about the application of a set of solutions to the overall sharing problem. This goodness revision has the additional advantage of being able to cope with external network changes or the derived concept-drift effect of changes in the the relationship among state and action spaces.

$$g_{endSpread} = \begin{cases} g \cdot (1 + k_g) & \text{if } (t_{spread} < t_{avg}) \\ g \cdot (1 - k_g) & \text{otherwise} \end{cases} \tag{7.51}$$

Chapter 8

P2P sharing network Simulator

This chapter presents a P2P sharing network MAS Simulator. It has been developed as a tool to empirically test the 2-LAMA in the P2P scenario, which is specified in previous two chapters. The corresponding tests and results are detailed in next chapter.

The simulator provides several facilities to analyse system behaviour and compare different coordination approaches with the same initial conditions. In fact, its internal architecture is conceived as a set of extensible modules which facilitate testing and comparing different approaches. This chapter starts introducing its features, provides usage directions and finishes with a description of its implementation and extension capabilities.

8.1 Introduction

In order to test our approach in the P2P case study, we have implemented a *P2P MAS simulator* (P2PMASsim). This simulator provides different facilities to execute tests and analyse results. Moreover, as it simulates both agents and network components, it allows to execute different approaches with identical initial conditions.

In fact, the simulator's base code can be used to develop different MAS where agent actions are solely illocutions —i.e. sending messages. Furthermore, it simulates a packet switching network simulator to transport agent messages —see §6.3. Accordingly, agent communications latencies depend on this network topology and its traffic. In other words, P2PMASsim can be used to perform simulations of MAS where agents interact through a communications network, like in the P2P case study.

In fact, we use this base code to implement the 2-LAMA specification described in chapters 6-7. Even more, we also implemented the BitTorrent protocol detailed in §6.2, as a base-line approach. These implementations let us perform

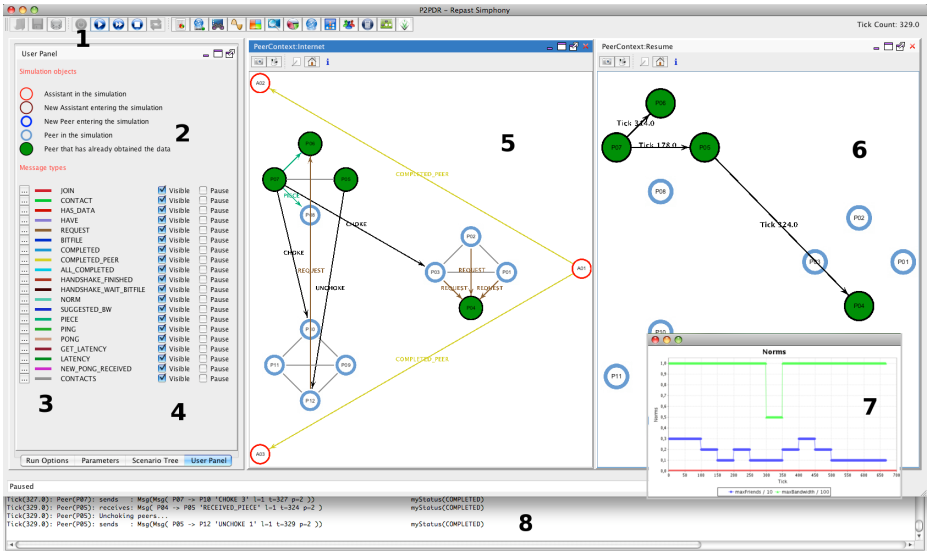


Figure 8.1: Simulator's basic view.

the empirical evaluation of our 2-LAMA approach discussed in next chapter 9. Since P2PMASsim simulates the communication network, it can guarantee the same network initial conditions to all experiments —also, it prevents any traffic beyond the generated by the MAS activity if it is not introduced on purpose. Even more, it generates relevant statistical information about all network components performance.

Above all, P2PMASsim provides control over all involved components and access to its activity that lets analyse system behaviour. As an illustration, figure 8.1 shows and screen-shot of the simulator to show its general features.

On top of the figure, under the title bar, there is a *Control toolbar* (1) that allows, among other features, to play the simulation, pause it or execute it step by step. Besides, it also shows current tick counter, since it is a discrete-event simulator.

On the left area, there is a *Legend panel* which shows information about what represents each layout object (2), the colour in which each sort of message among agents is displayed (3), whether messages are visible or not, and if execution will pause upon sending some of them (4). All these options can be modified by user. Thus, the legend allows an easy identification of each object and message to interpret what is happening in the simulation at every tick.

On the centre, there is the *Main layout* (5) that shows the elements of the simulation and the communications among them. On the one hand, regular agents and assistant agents are drawn according to its logical cluster topology —e.g. there are three clusters in the figure. On the other hand, messages are displayed as arrows among them with the corresponding colour defined in the legend panel.

Table 8.1: *Setup* panel parameters.

Keyword	*Description*
`Problem`	File describing scenario (peers, assistants, datum location).
`Net. topology`	File describing net topology (terminations, links, routers).
`Coord. model`	Coordination Model: BT, 2L-S, 2L-S-N-Heu, 2L-S-N-CBR.
`Data location`	Which peers have the datum initially.
`Time limit`	Max. simulation steps (sim. is aborted when exceeded).
`Norms`	Initial norm values (max_{BW}, max_{FR}, max_{HAS}).
`Adapt. int.`	Time steps elapsed among norm adaptations ($adapt_{interv}$).
`Reuse cases`	Reuse existing CBR case bases or start with empty ones.
`Sim. weights`	Weights used when comparing CBR cases (w_i^{Θ}, see §7.5.3.1).

On the right, there is a *Summary layout* (6) that displays how data has been distributed among different peers. It highlights completed peers and displays arrows connecting source and receiver agents. These arrows are labelled with the time step at which the datum was received. Moreover, the simulator can plot the evolution of different parameters along simulation, such as norm updates (7).

In addition, P2PMASsim generates a log file that contains all events occurred during an execution (8). Furthermore, the simulator includes a module for facilitating the analysis of these events. For this purpose, this module processes the generated logs extracting relevant information, which can be processed later. For instance, it can be used to compare the time spent to share data in different configurations, or using different sharing methods.

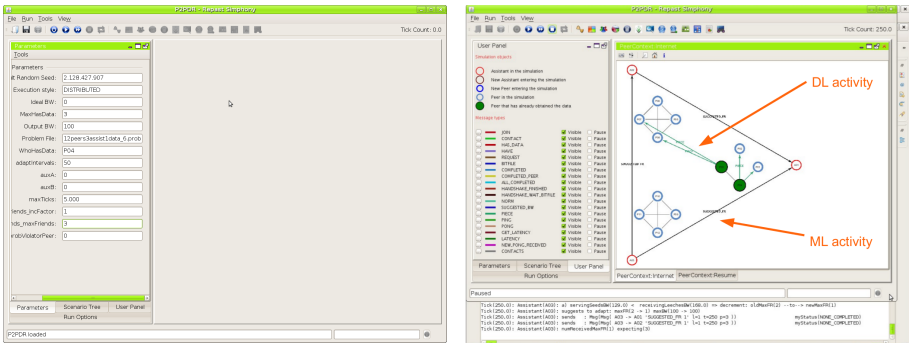
Next sections are devoted to provide further details on the simulator characteristics and its extensible architecture.

8.2 Usage and features

The basic cycle when running a simulation in is composed by three main stages: the setup, the execution, and the end of a simulation. Next subsections describe these stages and include some figures to illustrate some simulator's features. For the sake of simplicity, the examples refer to an interactive session although the simulator can be also run in batch mode. The former (interactive session) is useful to understand a single execution, whereas the latter (batch mode) is more appropriate for comparing several executions.

8.2.1 Setup of a simulation

First, when the simulator is loaded, it offers the *Setup* panel depicted in figure 8.2a. This panel lets set the execution options that are listed in table 8.1. Depending on these options, the simulation is executed using different approaches with different parameters.



(a) Initial Setup panel. (b) A running simulation.

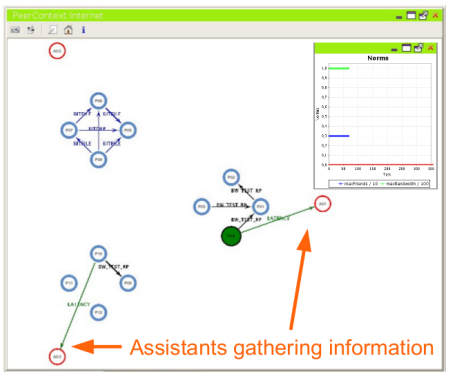
Figure 8.2: Simulator initialisation.

On the one hand, the simulator currently supports the following coordination models (labelled `Coord.Model` in table 8.1): BitTorrent (BT, see §6.2), 2-LAMA with social structure adaptation (2L-S, see §7.2), 2-LAMA with social structure and norm adaptation using the heuristic (2L-S-N-Heu, see §7.4), and 2-LAMA with social structure and norm adaptation using CBR (2L-S-N-CBR, see §7.5). On the other hand, the parameters include the name of two files: one that describes the agents of the system (`Problem`, see §6.4), and another one that defines the underlying communications network (`Net. topology`, see §6.3).

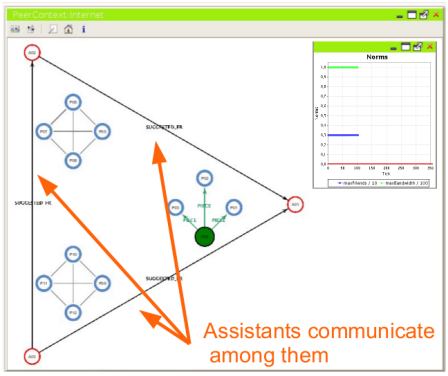
Although, the `Problem` file defines default values for all the execution parameters, the Setup panel lets change their values to perform a particular execution without updating external files. For instance, there is a parameter that lets the user change datum initial position (`Data location`). Also, the panel lets update the initial norm parameters (`Norms`) as well. These norm parameters are used by all 2-LAMA approaches, and some of them (2L-S-N-Heu, 2L-S-N-CBR) adapt their values at an adaptation interval (`Adapt. int.`) also defined in this panel.

Moreover, there is a parameter (`Reuse cases`) that defines if assistants, in the CBR approach, load the cases collected by previous execution or they start with an empty case-base —see §7.5.3. The weights used when computing the similarity among two cases (`Sim. weights`) in this CBR approach, can also be specified in this panel —see §7.5.3.1.

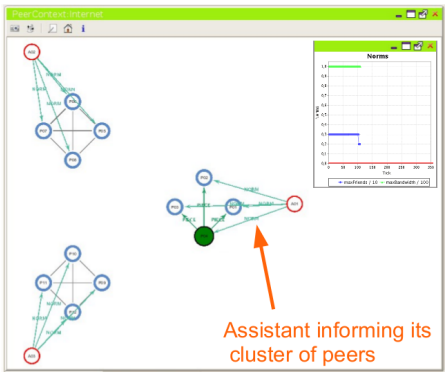
Once these parameters are defined, the user can start the simulation by using the *Control toolbar* (see figure 8.1-1). Then, the Setup panel is replaced by the *Legend panel* and the *Main layout* as shown in figure 8.2b —see previous section. This layout draws both meta-level agents (circles in red colour) and domain-level agents (in blue colour, or in green colour if they are completed) grouped by clusters —e.g. there are three clusters in figure 8.2b. The activity of these agents is represented by textual descriptions that are saved in a log file and illustrated by arrows among them, which indicate the sort of message they exchange.



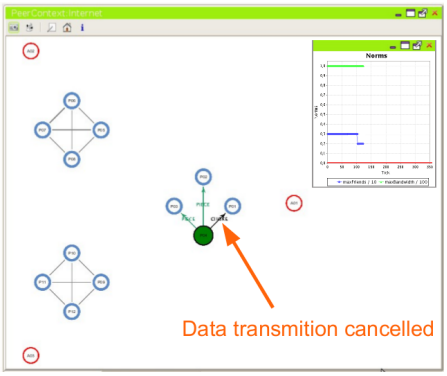
(a) ML gathering local information.



(b) ML agreement.



(c) ML updates DL norms.



(d) New DL activity.

Figure 8.3: Example of 2-LAMA adaptation steps.

8.2.2 Execution of a simulation

As stated, the user can command a simulation by controlling its time steps using the *Control toolbar* (see figure 8.1-1). Essentially, the user can run it step by step, it can pause the simulation at a certain time step or it can define a real time delay for each simulated time step. Moreover, the simulator is also able to pause simulation when certain messages are sent (see the Legend panel in §8.1). This provides a great tool to pay attention to certain agent interaction phases.

The main tool to visualise the system evolution along the execution is the *Main layout* (see figure 8.1-5) which depicts the messages exchanged by agents. Besides, the *Summary layout* (see figure 8.1-6) is also updated along the sharing process to show which agents are completed (drawn in green colour), at what tick they received the datum and from which participant. In addition, it is also possible to inspect environment and agent properties by using the GUI, and even to plot some of them (e.g. norm parameter values). As an illustration, figure 8.3

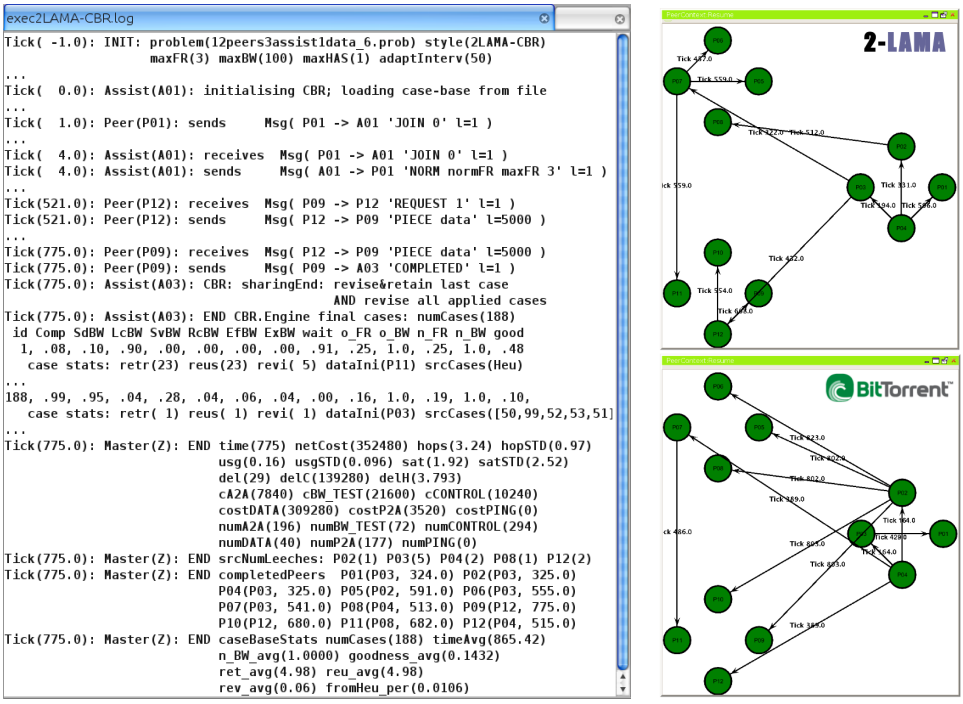
shows the Main layout and a plot of norms evolution along a 2-LAMA adaptation process —see §7.3. First, assistants collect local information about their cluster as illustrated in figure 8.3a. Next, assistants take their local decision and send it to other assistants as part of their agreement process. This message exchange is depicted in figure 8.3b. As a result, each assistant computes the new norms and it informs those participant agents in the same cluster. In figure 8.3c, the Main layout shows how assistants send the corresponding message to their agents. Also, the figure contains a superimposed plot about norms which reflects that the maximum number of simultaneous served agents (max_{FR}) is decremented —it becomes 2 instead of 3. Finally, the domain-level activity is influenced by this new norm as illustrated in figure 8.3d. In the example, the agent that was sending the datum to three other agents, cancels one of these transmissions to comply with the new norm.

In addition to described graphical information, all messages and predefined events are stored in a log text file. Figure 8.4a has some pieces of such file in order to illustrate its contents. Notice that it is composed by textual entries that describe different events indicating at which tick they happen. The most of these entries are generated by the simulator's infrastructure depending on the debug level —e.g. sending or receiving messages. In addition, the agents themselves can also add their own entries depending on their implementation.

8.2.3 End of a simulation

At the end of a simulation, there is different information is provided as an execution summary. On the one hand, the *Summary layout* is an illustration of how the datum was distributed among participants. It visually shows who provided the datum to each agent (as an arrow) and at which tick the destination agent received it (as labels over arrows). This information lets the user perform a visual comparison among different executions. For instance, figure 8.4b shows the summary layout of two executions that use a different sharing method —the top part corresponds to an execution using 2-LAMA, whereas the bottom part corresponds to the base-line BitTorrent.

On the other hand, there is statistical information added to the mentioned *log file* as illustrated in figure 8.4a —specifically, those messages which contain the 'END' keyword. Table 8.2 provides a short description of them. The most relevant metric is the time invested in the sharing process (`time`) which is used to evaluate system's performance —see §6.5.4. Besides, there are some network metrics (e.g. `netCost`, `hops`, `usg`, `sat`) that summarise net activity —see 6.3.2. Moreover, the number of messages and its network cost is detailed per each type of message (`num<type>`, `cost<type>`). Also, these details contain information about the network consumed by those messages that were cancelled (`del`) before achieving its destination —see §6.7.3. Besides, there is also a summary of how data was spread among agents. Specifically, there is the number of incomplete agents served by each complete agent (`srcNumLeeches`) and which agent provided the datum to each participant at a certain tick (`completedPeers`). Finally, if the execution used the CBR approach, there is a set of statistical information



(a) Activity log file. (b) Visual summaries.

Figure 8.4: End of simulation.

about it (`caseBaseStats`). It comprises, among others, the final number of cases in the case-base (`numCases`) and the average of the number of times the cases were retrieved, reused and revised (`ret_avg, reu_avg, rev_avg`). Even more, assistants print their case-base, which includes this sort of statistical details for each particular case.

Overall, once a user has used the interactive GUI to understand the behaviour of a given approach, he can evaluate and compare it by just using logged information. In fact, the simulator allows the execution of multiple simulations with different run options in batch mode, generating a log file for each simulation.

In particular, P2PMASsim provides analysis facilities to process data from more than one log at once (multiple-simulation). Thus, it allows to summarise and compare the performance and behaviour of different simulations. As an illustration, the average of sharing time when data is initially in different peers would conform a *multiple-simulation summary*. Moreover, plotting this average time for two different approaches with different initial norm parameters as shown in figure 8.5 is an example of a *multiple-simulation comparison*. Such figure shows two plots comparing the performance (time in ordinate) among different 2-LAMA approaches (left: without norm adaptation, right: with norm

Table 8.2: Simulator's available metrics.

Keyword	*Description*
`time`	Time required to spread datum among all peers.
`netCost`	Total network cost.
`hops*`	Average number of links traversed by each message.
`usg*`	Ratio of links' bandwidth used at each time unit.
`sat*`	Ratio of data units that cannot traverse links because there is saturation.
`A2A`	Number and average cost of messages among assistants.
`P2A`	Number and average cost of messages among participants and assistants.
`num<type> / cost<type>`	Number and average cost detailed by each type of message among domain-level agents (`BW_TEST, CONTROL, DATA`).
`del`	Number, average cost and hops of cancelled data messages.
`srcNumLeeches / completedPeers`	Measures about how data was actually distributed among domain-level agents.
`caseBaseStats`	Statistics about the CBR.
* ⇒	these measures are an average on all network links; their standard deviation is also available (with STD suffix).

adaptation), and different initial norm parameters (max_{FR} in abscissa, max_{BW} in series and max_{HAS} fixed to 1).

This turns out to be very useful for system designers, since rather than just knowing the overall system performance, it helps to understand its evolution depending on different parameters and alternatives. As a consequence, if some problem arises, it is easier to identify it, its possible causes and what is most valuable: which simulation parameter values perform best.

8.3 Extensible Architecture

The design of the P2PMASsim is conceived as an extensible architecture to develop and test different MAS approaches in the P2P sharing network scenario. It has an internal architecture that clearly isolates different functionalities and contributes to its extension capabilities. For instance, it is possible to use all implemented code and simply change an adaptation function or it is possible to reuse only the network simulation layer in other scenarios where agents communicate through a packet switching net. Next subsections detail the initial requirements and its architectural extensible design.

8.3.1 Requirements

The initial *requirements* to develop P2PMASsim comprised:

- to have a discrete-event simulator.

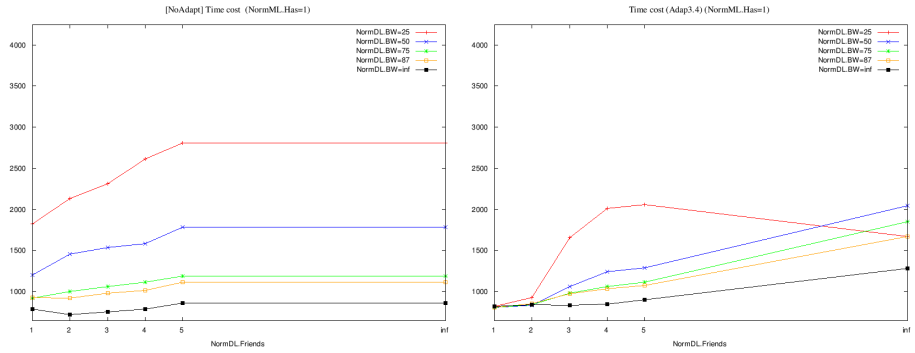


Figure 8.5: Multiple-simulation comparison example: time versus norm parameters and approaches.

- to have facilities to simulate agents.
- to simulate the communications network.
- to visualise system activity in two levels (like 2-LAMA model).

In particular, working with a simulator allows to initially compare different MAS approach concepts in the P2P scenario without need to deploy real computers attached to different network configurations. Moreover, a simulator allows having control and access to all system components. This contributes to guarantee a fair comparison among the tested approaches.

Furthermore, having a discrete-event simulator simplifies agent development, since it avoids concurrent programming issues. This simplification is also enhanced by counting on simulator tools that help to implement agents whose acts are restricted to exchanging messages.

Besides, simulating the communication network contributes to have the mentioned control and access to all system components. In fact, this lets implement exactly the simplified packet switching model described in §6.3. Otherwise, using an existing network simulator requires to set several low level network details.

In addition, a visualisation close to 2-LAMA model should help to analyse such approach. This visualisation should distinguish among domain-level and meta-level agents, show them grouped by clusters, and present their relevant interactions —i.e. the messages they exchange.

8.3.2 Design

In order to meet the specified requirements, the simulator is implemented in Repast Simphony [North et al., 2005]. Briefly, Repast is a simulator framework which provides the following features:

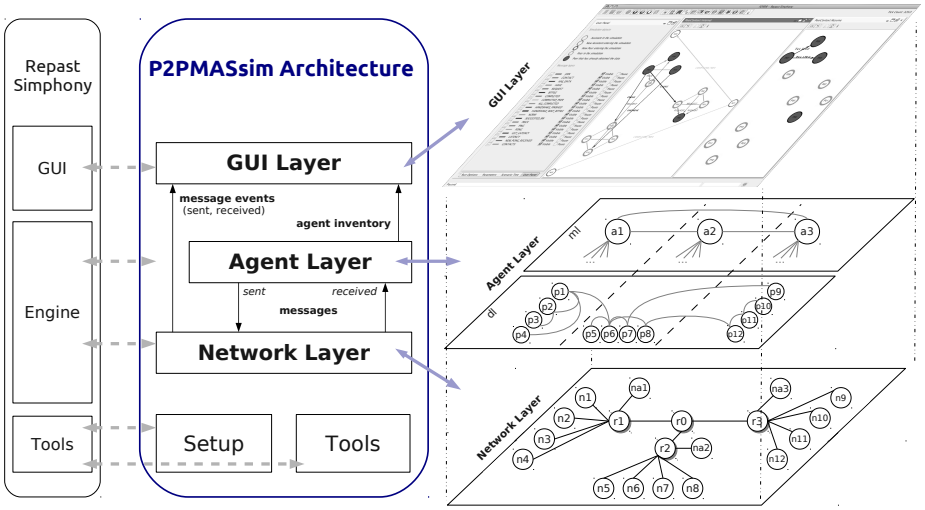


Figure 8.6: Simulator's architecture overview.

1. it provides *a basic Graphical User Interface* (GUI) with a generic layout and simulation step control.
2. it has a *discrete-event scheduler* that can call methods at certain ticks.
3. it allows to run simulations in *batch mode*.

These characteristics constitute a base-line but they are not enough to achieve the requirements enumerated in previous subsection. As a consequence, our P2PMASsim code adds the following features to achieve the requirements:

1. An *extended GUI* that shows the activity of two level of agents grouped by clusters, visualises some of the messages they exchange, and presents a summary of the sharing process.
2. A basis to develop *agents that follow a certain communication protocol* —e.g. state machine based agents that act by sending messages.
3. The *simulation of a packet switching network* to transport agent messages.
4. A report of simulator activity into *log text files*, and the corresponding tools to analyse them.

In particular, P2PMASsim provides these features through the layered architecture depicted in figure 8.6. In short, it has a component for each previous mentioned feature (GUI layer for feature 1, Agent layer for 2, Network layer for 3, and Tools for 4) and a Setup component to arrange all them. As stated, this architecture clearly isolates different functionalities. Moreover, these layers are loosely coupled since their interaction is based on agent message exchanges.

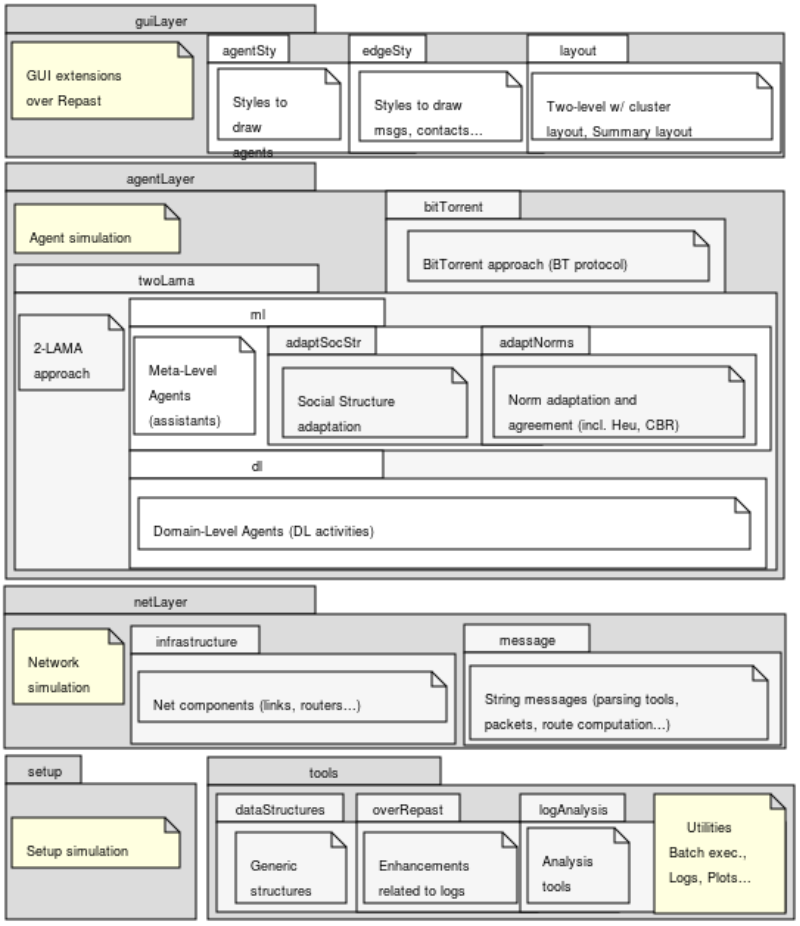


Figure 8.7: Simulator's components.

This feature makes it easier to update each layer independently or even to use them separately in other domains.

Figure 8.7 presents a summary of each component which are detailed in subsequent sections. These sections have some diagrams about the internal structure of these components. These diagrams are *simplifications* of Universal Modelling Language (UML) *class diagrams* since they just aim to illustrate the general structure of such components —e.g. they neither contain all classes nor all class members. In fact, P2PMASsim is compound of more than 50,000 lines of code belonging to 265 classes, and next diagrams just show less than 25% of them.

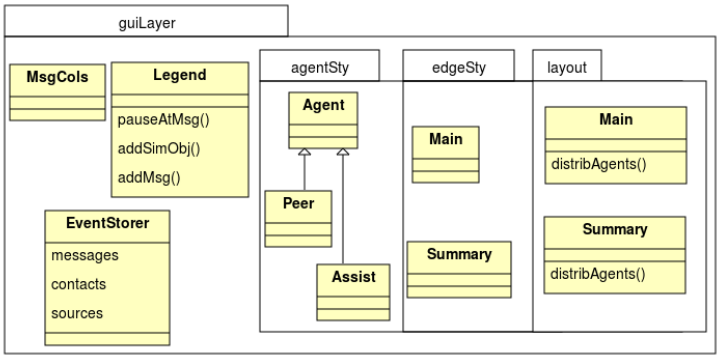


Figure 8.8: Simulator's GUI layer.

8.3.3 GUI layer

The Graphical User Interface (GUI) layer of P2PMASsim extends the basic features of Repast GUI. Figure 8.8 shows its components. In short, there are two *layout* subcomponents: `Main` and `Summary`. The former provides the *Main layout* depicted in figure 8.1-5, and the latter provides the *Summary layout* shown in figure 8.1-6. In both cases, the corresponding classes offer a method (`distribAgents`) which is in charge of distributing agent representation in the corresponding canvas[1]. This method distributes meta-level agents around a single external ring, and the domain-level agents are distributed in several inner rings that correspond to clusters —see figure 8.1.

The agents are drawn according to previous distribution by using the agent styles (`agentSty`) which extends Repast basic agent representations. Analogously, the edges among agents —which represent messages in the Main layout, and datum transmissions in the Summary layout– are extensions (`edgeSty`) to Repast basic edge representations too. The colours of these edges are stored in a container (`MsgCols`) and can be updated from the *Legend Panel* depicted in figure 8.1-3. This panel is managed by a class (`Legend`) that lets define which messages may pause the simulations.

Precisely, the interface among the GUI layer and the other layers relies only on message events. In particular, the Network layer notifies the GUI layer every time a message is sent or received. This simplifies components and agent implementation, since they do not need to manage the GUI. On the contrary, it is the GUI layer which extracts all its information from message contents. Specifically, the GUI layer has a subcomponent that receives all these message notifications (`EventStorer`) and extracts the relevant information that is used by layout classes. For instance, it identifies that two agents have a contact relationship if they exchanged `bitfield` messages —see §6.2.1 and §6.7.2. Then, this fact is stored in an attribute (`contacts`) that is checked by the Main layout

[1] The word *canvas* is used to denote the window's region where the graphical representation is drawn.

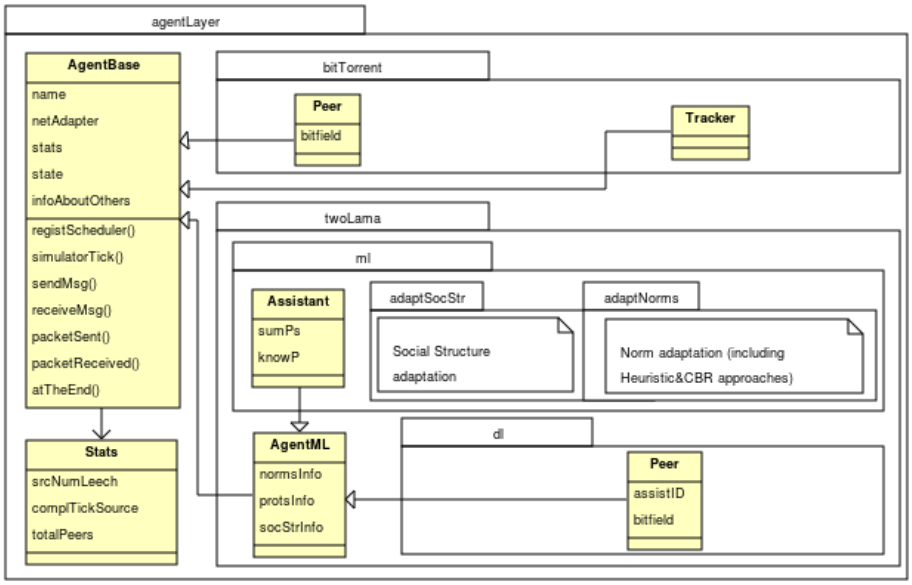


Figure 8.9: Simulator's Agent Layer.

in order to draw a grew line among those two agents. Similarly, when an agent in sending the datum to another one, this relationship and the tick when the datum was totally transmitted is stored in another attribute (`sources`). In this case, this information is used by the Summary layout to draw the corresponding labelled line.

Overall, these mentioned classes can be easily updated, extended or replaced to obtain other layout distributions, entity representations or traced events. This can be useful to work with other approaches on the same P2P sharing network scenario, but also to work with other scenarios since the code is encapsulated and only relies on message events.

8.3.4 Agent Layer

The Agent layer of P2PMASsim provides the basis to develop agents that follow a certain communication protocol, but also particular agent implementations depending on the BitTorrent and 2-LAMA approaches —see chapter 6. Figure 8.9 depicts its components.

8.3.4.1 Agent base and statistics

The main component is an agent code skeleton (`AgentBase`) that provides a state machine foundation. In short, it has an attribute which contains its current state (`state`) —e.g. joining the system, looking for the datum or serving the datum. Also, it has an attribute (`infoAboutOthers`) where it stores information about

other agents, such as the relative protocol state —e.g. handshaking, receiving datum or choked.

This agent base code, has a reference to the network adapter (`netAdapter`) it uses to send messages to other agents. This adapter belongs to the Network layer, which will transport these messages. In particular, `AgentBase` has a private method (`sengMsg`) that is used to send string messages. This is done by just indicating the name of the target agent since the Network layer is in charge of locating the network termination where the message should be delivered.

The implemented agent can be proactive and/or reactive. On the one hand, it can initiate different actions from a method which is automatically scheduled to be called at each tick (`simulatorTick`). On the other hand, it can react to any received message since its network adapter triggers a callback method (`receiveMsg`). Furthermore, the network adapter also triggers a callback method (`packetReceived`) when a packet is received —e.g. a datum message requires 500 packets, so `packetReceived` is called each time, whereas `receiveMsg` is called only once when all packets have been received.

Besides, the agent base has a reference to a class that contains statistical information about the agent layer (`Stats`). It follows a singleton pattern, so there is a single instance of this statistical class. This way, all agents can access this single object in order to notify certain events. For instance, when an agent receives the datum from a certain participant, it notifies this fact to the this statistics object. Accordingly, there is an statistic attribute (`complTickSource`) which stores that a particular agent received the datum from a particular source at a certain tick. Moreover, another statistic attribute (`srcNumLeeches`) stores that the source agent has sent the datum to an additional participant. This sort of information is retrieved at the end of the simulation (`atTheEnd`) when printing the statistical information to the log file —see §8.2.3.

All in all, these classes can be easily reused to develop agents in other scenarios or they can also be extended to collect other sort of statistics.

8.3.4.2 Coordination model implementations

Currently, the simulator includes implementations of agents according to BitTorrent and 2-LAMA. These agents extend the base code described in previous section. The right part of figure 8.9 depicts the corresponding subcomponents.

On the one hand, the BitTorrent component (`bitTorrent`) includes an agent specialisation for the *peer* role (`Peer`) and another one for the tracker role (`Tracker`) —see §6.2. In both cases, these implementations react to each BitTorrent protocol message when they receive it.

On the other hand, the 2-LAMA component (`twoLama`) includes two subcomponents, one for the meta-level (`ml`) and another one for the domain-level (`dl`). Their agents derive from a common code skeleton (`AgentML`) which provides them with their individual knowledge about norms (`normInfo`), the protocol (`protsInfo`), and the social structure (`socStrInfo`) —e.g. they are containers of current norm definitions or current contacts. Additionally, the domain-level agent (`Peer`) keeps the identifier of its corresponding assistant and its bitfield,

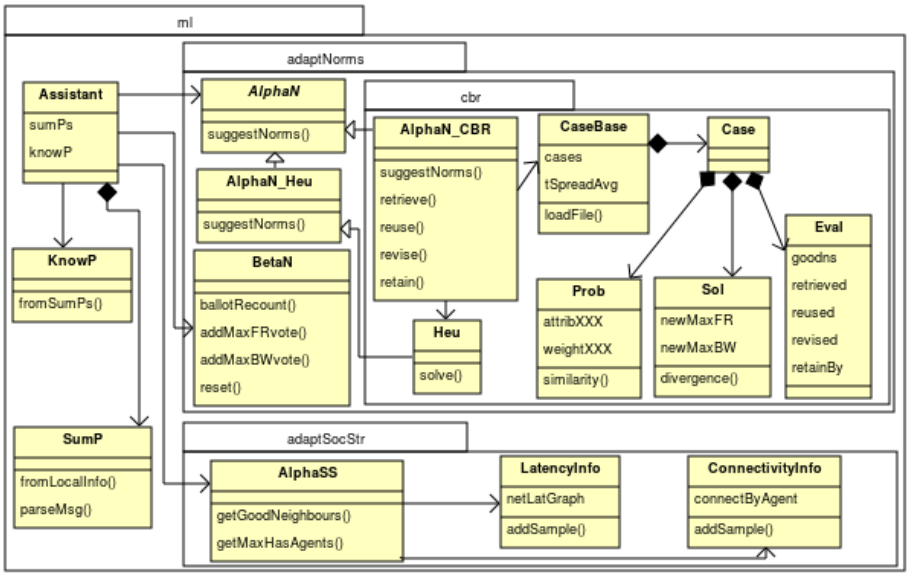


Figure 8.10: Simulator's Meta-Level in 2-LAMA approach.

which describes whether it has the datum or not —see §6.5. That is to say, in current implementation domain-level agents already know to which assistant they should contact —i.e. they know to which cluster they belong to. Alternatively, they could perform an initial bandwidth test –see §6.7.2– to measure their communication latency to each assistant and choose the closer one.

Besides, figure 8.10 provides further details about the meta-level implementation. Its agents (`Assistant`) contain the summary (`SumP`) created from their local information (`fromLocalInfo`) and the summaries received from other assistants (`parseMsg`) —see §5.5.3.1. Both are required when creating its knowledge information (`KnowP`) from summaries (`fromSumPs`). Then, this knowledge is used when calling the corresponding norm adaptation function approach (`suggestNorms` in `AlphaN_Heu` and `AlphaN_CBR`). Notice, that there is an interface (`AlphaN`) devoted to homogenize current and future implementations of different adaptation approaches.

Current implementations of the norm adaptation function are the heuristic approach (`AlphaN_Heu`) and the CBR approach (`AlphaN_CBR`) —see §7.3.2. The `suggestNorms` method of the heuristic approach implements the algorithm 7.1, whereas the `suggestNorms` method of the CBR approach implements the algorithm 7.3. The latter has other methods which implement the CBR cycle steps described in §7.5.3 and uses an additional class (`Heu`) which extracts its information from a problem definition, uses the heuristic approach (`AlphaN_Heu`) and returns its results as a CBR case.

In addition, the CBR approach requires a case-base (`CaseBase`) which in-

cludes a collection of cases (`cases`) that can be loaded form a file (`loadFile`) to keep the case-base among executions. Also, it keeps the average of t_{spread} (`tSpreadAvg`) which is needed to evaluate cases at the end of a sharing process —see §7.5.3.4.

The implementation of a case is a class (`Case`) that aggregates other classes which contain the problem description (`Prob`), its solution description (`Sol`) and its evaluation (`Eval`). The rational of such implementation is to have more flexibility when testing different approaches from a research point of view —instead of operating with a fixed approach from a performance point of view. For instance, it has been useful to test different problem attributes (`attribXXX` stands for a class member for each attribute), different similarity functions (`similarity`, currently it implements eq. 7.41), different divergence functions (`divergence`, see δ in §7.5.3.2), different evaluations (`goodns`, see §§7.5.3.3-7.5.3.4). Besides, the `Eval` component also collects statistical information for each case. For instance, the number of times the case was retrieved (`retrieved`) or which agent has initially the datum in the execution that added the case (`retainBy`). This statistical information is printed to the log file at the end of the execution —see §8.2.3.

Further, there is a class (`BetaN`) related to the agreement function — see §7.3.3.1. In particular, it has a method (`ballotRecount`) that computes new norm parameter values according to collected votes (through methods `addMaxFRvote` and `addMaxBWvote`).

Similarly to the norm adaptation function, the social structure adaptation function is also encapsulated in a class (`AlphaSS`). Current implementation has two methods to perform the computation described in §7.2.2. One method (`getGoodNeighbours`) returns the contacts of a domain-level agent according to collected latency measures (`LatencyInfo`). Whereas the other method (`getMasHasAgents`) returns the max_{HAS} agents which have a better connectivity according to collected measures (`ConnectivityInfo`). In particular, in current implementation latencies are computed like described in eq. 7.1 and stored as arc weights in a graph among participants (`netLatGraph`). Besides, current connectivity implementation is the one described in eq. 7.2 which is stored in a dictionary (`connectByAgent`).

Overall, this architecture facilitates testing different approaches for each component, simply by overloading or replacing certain classes. For instance, it is easy to add another machine learning adaptation method as a class that inherits `AlphaN`, or simply change how the cases similarity is computed by overloading the `divergence` method in `Prob` class. Furthermore, it is possible to add new sorts of agent by extending current code skeletons.

8.3.5 Network Layer

The Network layer of P2PMASsim transports agent messages by simulating the packet switching network described in §6.3. This way, the messages are not received instantaneously but they are delayed depending on network topology, net status and message length. This behaviour is suitable to simulate the P2P sharing network scenario, but it can be also useful in other domains that require

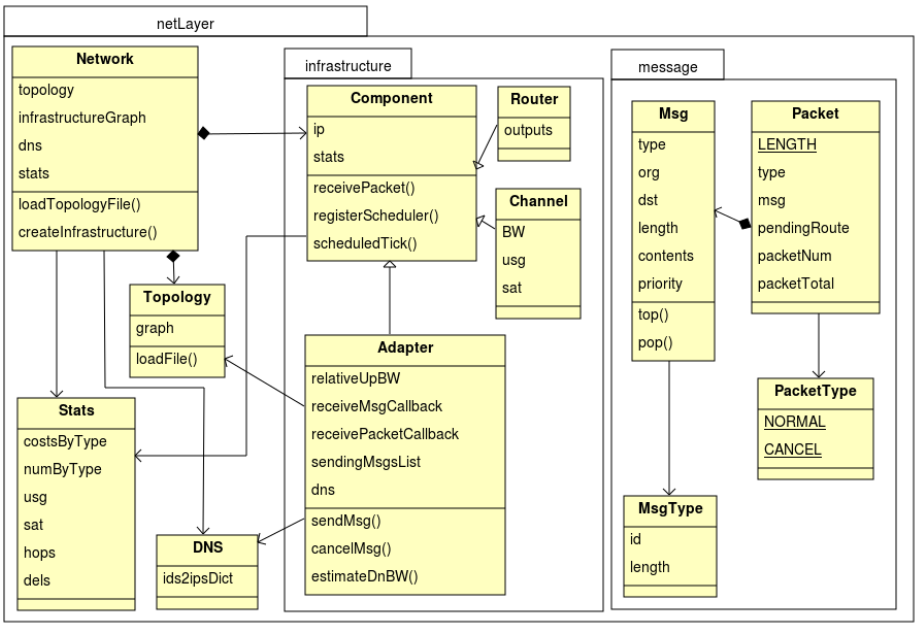


Figure 8.11: Simulator's Network layer.

this sort of communication network. Figure 8.11 depicts the internal components of this layer, which are described as follows.

8.3.5.1 Initialising the infrastructure

The main component is the network class (`Network`) which is able to load a network description from a file (`loadTopologyFile`). When loading this network description, it first loads a topology description (`Topology`) from a file (`loadFile`), which is stored as a graph (`graph`). The nodes of this graph describe routers or network terminations, whereas the edges describe the links among them — see §6.3.1. Next, the network class also initialises the network statistics singleton (`Stats`) and creates an infrastructure (`createInfrastructure`) composed by class instances that simulate each component (`Component`) according to the described topology —notice that a link is equivalent to both an upload channel and a download channel. These class instances will simulate each network component when they receive data as detailed below. Notice that each component has a reference to the statistics object (`stats`), so it can notify certain events to update the statistical information. This information comprises the metrics defined in §6.3.2 that are shown at the end of the sharing process as described in §8.2.3.

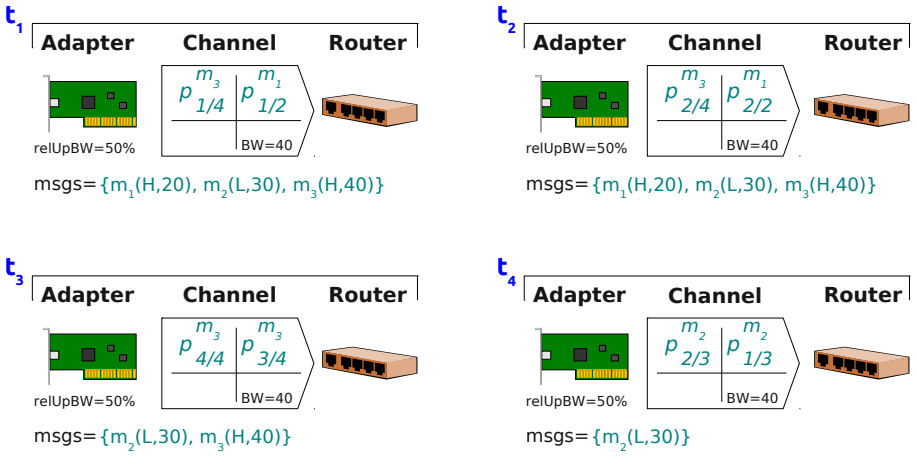


Figure 8.12: Example of an adapter sending packets.

8.3.5.2 Exchanging messages

The created network is used by agents to exchange messages. In particular, previous section 8.3.4.1 mentioned that each agent has a reference to a network adapter (`Adapter`). This means that each agent is attached to a network termination, with a given network address (`ip`). When the agent wants to send a message, it injects it into its network adapter (`sendMsg`). Then, this message is split into packets that travel along links and follow their path by switching at routers depending on network topology. The latency of a packet from a network adaptor to another one depends on the number of links, their bandwidth and the current traffic through them. When packets achieve the destination point, the adapter calls the agent method registered to receive a packet (`receivePacketCallback`). Eventually, if it is the last packet of a message, the adapter also calls the agent method registered to receive a whole message (`receiveMsgCallback`). This way, agent implementations can pay attention to packets or just wait for entire messages.

In more detail, a message (`Msg`) contains its priority (`priority`), its contents as a string (`contents`), the identifier of the receiver agent (`dst`), and the message type (`MsgType`) which may determine its length (`length`). Hence, when an adapter receives a message to inject into the network, it looks for receiver's network address using a dictionary (`DNS`) that translates agent identifiers to network addresses (ip). Then, the adapter adds this message to its internal list of messages being sent (`sendingMsgsList`).

As a consequence, at each tick, adapter's main method (`scheduledTick`) creates packets (`Packet`) for each message being sent. The generated packets depend on upload channel's bandwidth (`Channel.BW`), the amount of bandwidth the agent wants to use (`Adapter.relativeUpBW`), the current messages (`Adapter.sendingMsgsList`) and their priority (`Msg.priority`). Figure 8.12 illus-

trates this behaviour with an example. In the example, an agent has an upload channel with `BW=40` data units per tick. However, it sets a `relativeUpBW=50%` in order to fulfil a given $max_{BW} = 50$. Accordingly, its adapter should send 40 data units at each tick. Besides, the established packet size is `Packet.LENGTH=10`, there are three messages to sent `sendingMsgsList=`$\{m_1, m_2, m_3\}$, their lengths are $\{20, 30, 40\}$ data units respectively, and their priorities are $\{high, low, high\}$ respectively —in top left part of figure 8.12, 'H' stands for high priority and 'L' for low priority. Then, at tick t_1, the adapter creates and injects two packets, the first out of two of m_1 and the first out of four of m_3 —because m_1 and m_3 have a higher priority than m_2. Next, at t_2 the adapter injects the last packet of m_1 and the second of m_3. Hence, at t_3 , m_1 is already sent, and the adapter generates the last two packets of m_3. Finally, at t_4, the adapter can start sending messages of the low priority message m_2.

When a channel (`Channel`) receives a packet, it delivers the packet to the subsequent router (`Router`) –or network adapter– in the next tick. Then, the router is in charge of injecting the packet to the corresponding channel depending on its path (`pendingRoute`). In real networks, routers have a buffer to store packets if they cannot be sent through a given channel due to its bandwidth capacity. However, in current implementation this buffer functionality is provided by the channels themselves, which only let traverse a number of packets according to its bandwidth and store the rest of the packets to deliver them in subsequent ticks.

In order to behave as described, a component (i.e. an adapter, a channel or a router) uses two execution policies: a reactive policy on incoming packets and a scheduled policy on outgoing packets. For instance, no code is executed while the component is waiting for a packet. Then, when another component delivers it a packet, the code to handle incoming packets (`receivePacket`) is executed —notice that the way to deliver a packet to a component is to call this method. Such execution is performed in a tick. However, if further actions are required in subsequent ticks, the component itself registers into Repast scheduler (`registerScheduler`) a method (`scheduledTick`) to be called at next tick. This way, the scheduler calls the component at each tick while it has actions to perform —e.g. an adapter injecting packets as depicted in figure 8.12 or a channel delivering some of the packets it has in its buffer.

8.3.6 Other components

In addition to the three layers described above, P2PMASsim has some more components to support the layer components which are described in the following subsections.

8.3.6.1 Setup

On the one hand, there is the *setup* component which is in charge of configuring and managing them. Figure 8.13 depicts its subcomponents. It has

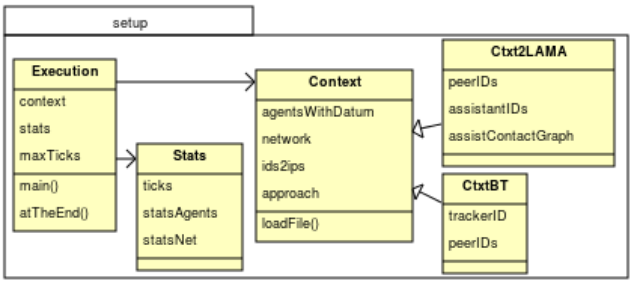


Figure 8.13: Simulator's Setup component.

a class (`Execution`) with the simulator's code entry point (`main`) that configures all layers depending on current context. In particular, a context definition (`Context`) is the description of the agent and network components that are simulated. For instance, it specifies which agents have initially the datum (`agentsWithDatum`), how is the underlying network (`network`) or where agents are attached to the network (`ids2ips`, where `ids` stands for agent identifiers and `ips` for network nodes identifiers). In fact, there are specialisations of this context class, which contain further details depending on the particular approach (`approach`). For example, the context description (`Ctxt2LAMA`) when running a 2-LAMA approach (see §6.4) provides a list of domain-level identifiers (`peerIDs`), meta-level agent identifiers (`assistantIDs`) and a meta-level social structure description (`assistContactGraph`, i.e. the net of relationships among assistants). Besides, the context description (`CtxtBT`) when running a BT approach (see §6.2) just provides a list of sharing agents (`peerIDs`) and the identifier of their tracker (`tracker`).

When configuring previous layers, the prime class (`Execution`) also creates the global statistics (`Stats`). They contain a reference to agent and network statistics (`statsAgents`, `statsNet`) and the number of simulated ticks (`ticks`). In fact, this prime class it the one in charge of printing the statistical information at the end of an execution (`atTheEnd`) —see §8.2.3. Moreover, this class has an attribute that defines the limit of simulated ticks (`maxTicks`) to prevent infinite executions. This limit is passed to Repast engine, in order to abort the execution if it is exceeded due to a coding error or an approach misconception.

8.3.6.2 Tools

On the other hand, there is a component (`tools`) which groups different utilities related to previous components. Figure 8.14 illustrates its subcomponents. It has a set of data structure implementations (`dataStructures`) which are used by different P2PMASsim components and could also be reused in other projects. For instance, it has a weighted graph implementation (`Graph`) which allows to attach a reference to any sort of information (`extra`) to each node (`Node`). This is used to attach network component type and identifier when using a graph to define

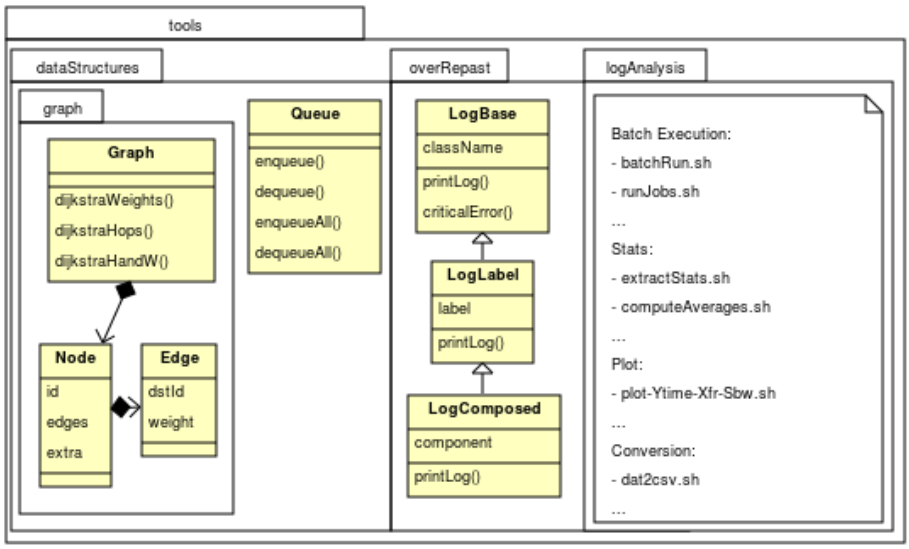


Figure 8.14: Simulator's Tools component.

networks' topology. Moreover, the graph implementation (`Graph`) offers methods to compute Dijkstra's algorithm [Dijkstra, 1959] on it. These methods obtain path costs from different parameters: from edge weights (`dijkstraWeights`), from the number of edges traversed (`dijkstraHops`) or both at the same time (`dijkstraHandW`, see [Campos et al., 2010]). These methods are used, for example, when computing the route of a packet along the communication network.

Besides, there is a set of classes which extend Repast log functionalities (`overRepast`) and could be used in other Repast projects. Almost all P2PMASsim classes derive from one of these log classes —e.g. all the classes mentioned in previous sections. The base one (`LogBase`) stores the name of the inherited class (`className`), which is added as a prefix to all printed messages. This way, when printing log messages from code (`printLog`), it is not necessary to include the name of the class which is generating the message, since it is automatically added —e.g. printing "Assistant" prefix to each log message printed by an assistant. Analogously, there is another class (`LogLabel`) that automatically adds an additional label (`label`) which can be used to print agent's identifier —e.g. printing the assistant identifier like "a2". Also, there is a third log class (`LogComposed`) which adds a component label (`component`) and can be used to automatically print an agent's internal component name —e.g. adding the "CBR" prefix when messages are issued from CBR classes. Besides, there are other methods related to log messages, such the one used when there is a critical error (`criticalError`). This method prints the log message with an additional "CRITICAL ERROR" prefix and stops simulator's execution.

Moreover, there is a set of tools (`logAnalysis`) focused on allowing batch

executions of several simulations with different parameters and analysing afterwards the corresponding results. Mainly, these tools are shell scripts, so they can easily run in remote server. This way it is easy to have different remote machines running different sets of simulations and collecting their results subsequently.

As an illustration, we describe a basic execution and analysis cycle. First, there is a script (`runJobs.sh`) which contains the definition of a set of simulations to execute. Hence, there can be different versions of this script in different remote machines in order to run a part of the total simulations in each computer. In fact, these scripts call several times to a basic script (`batchRun.sh`) that provides all necessary arrangements to perform a single simulation —e.g. it defines library paths or the Java Virtual Machine configuration. When a machine finishes its simulations, it can use a script (`extractStats.sh`) to extract desired statistic information from the generated log files. Moreover, the computer can run a script (`computeAverages.sh`) to compute different averages of these multiple-execution statistics.

Finally, these results can be collected in a single computer and use other scripts to plot this information. For example, figure 8.5 shows a chart generated with a script (`plot-Ytime-Xfr-Sbw.sh`) that draws the time (Y-axis) for different domain-level norms (max_{FR} parameter in X-axis and max_{BW} as series) when these norms are not adapted (left-part) or adapted by using the heuristic (right-part). These plots are generated using the standard Gnuplot tool [Janert, 2009], so they are fully configurable and can created remotely if necessary. In addition, it is also possible to convert data files into other formats in order to plot/process them using different software. For instance, there is a script (`dat2csv.sh`) to convert a data file into a comma separated values file, so it can be imported into a spreadsheet program to plot charts interactively.

8.4 Open MAS extensions

In addition to the features described in previous sections, P2PMASsim offers some extensions to simulate two open MAS issues: agents that enter and/or leave during the execution (Entering / Leaving agents) and non-compliant agents (Norm violations) —both issues are related to the lack of control over participant development by the system designer. Next subsections detail each extension by describing and motivating a possible underlying model and its current implementation.

In fact, the implementation of these extensions rises some complexities, specially the entering/leaving issue. For instance, when a source agent is leaving, the ones receiving the datum from it need to ask for the datum to other sources. This means that the agent code needs to be updated to let the agent enter or leave, but also to handle other agents entering/leaving. Even more, some of these changes imply enhancing the protocol itself and updating different simulator components.

In addition, due to the need for a fair comparison among different approaches, it is necessary to implement a mechanism that allows repeating a

similar sequence of violations or entering/leaving events among different simulations. The implementation of this mechanism rises more complexities again.

Overall, current conceptualisation and implementation of these extensions may be a guidance to explore other open MAS issues or enhance current ones. Furthermore, chapter 9 introduces some experiments using these extensions which provide further insights into analysing the behaviour of different approaches in an open MAS context.

8.4.1 Entering / Leaving agents

One characteristic of open MAS is that it is not feasible to control when agents enter or leave the system along the execution. For instance, in a real traffic scenario, cars may leave the street to enter a parking area or they may return to the street at any moment. Moreover, there can be different agent profiles, with different tendencies to perform such actions. For example, there may be agents leaving the system as soon as they get a profit, which we call *freerider* agents.

Furthermore, these entering/leaving actions may be voluntary –i.e. they depend on agent intentions– or forced by environmental circumstances —i.e. they do not depend on agent purposes. For instance, in a real P2P sharing network, connected computers may involuntarily leave the system due to a net failure. Hence, in contrast to voluntary actions, the involuntary actions may not leave agents the opportunity to notify their actions properly.

Accordingly, P2PMASsim entering/leaving model offers different parameters to control voluntary and involuntary actions, as described in next subsection. Besides, [Novo and Campos, 2010] provides further information about the model and its implementation.

8.4.1.1 Model

The P2PMASsim entering/leaving model is focused on changing initial MAS population at certain intervals. This process depends on a reference network topology (it indicates the proportion of agents connected to each cluster and their possible bandwidths) and a set of parameters (listed in table 8.3). Mainly, there are four sets of parameters to control: the amount of agents connected along the execution (General), the entering process (Enter), the involuntary leaving process (Network failure) and the voluntary leaving process (Voluntary).

These parameters are relevant in the following entering/leaving phases:

- (1) *Initial phase*: initially a percentage of agents (`INITIAL_PEERS`) over the maximum number of agents allowed (`MAX_PEERS`) is created. Their bandwidth and cluster is randomly selected from the reference network topology. A percentage (`PROB_PEER_FREERIDER`) of these created agents are *freeriders*, which means that they may leave the system easily —see the Voluntary leaving phase.

- (2) *Delay phase*: after the initial phase, the entering/leaving model is inactive until a given tick (`STARTING_TICK`). This way, the domain-level can

Parameter	*Val.*	*Description*
General		
MAX_PEERS	15	max. number of peers during the execution
INITIAL_PEERS	0.80	percentage of peers initially connected
MIN_PEERS_CONNECTED	0.60	percentage of the minimum remaining peers
STARTING_TICK	75	tick when the E/L system is active
MIN_LEECHES_TO_STOP_EL	2	E/L is inactive when there only this #leeches
Enter		
ENTERING_PEER_INTERVAL	100	ticks among entering steps
NUM_ENTERING_PEERS	0.20	percentage of possible entering peers
PROB_ENTERING_PEERS	0.50	probability to effectively enter the system
PROB_ENTERING_COMPLETED	0.25	probability that an entering peer is complete
Leave: Network failure		
PEER_FAILURE_INTERVAL	100	ticks among net failure leaving steps
POSSIBLE_NUM_PEER_FAILURES	0.20	percentage of possible leaving peers
PROB_PEER_FAILURE	0.50	probability to effectively leave the system
Leave: Voluntary		
LEAVING_PEER_INTERVAL	100	ticks among voluntary leaving steps
PROB_PEER_FREERIDER	0.50	probability of a peer to be a freerider
PROB_LEAVING_SEED_NORMAL	0.10	prob. to leave of a complete non-freerider
PROB_LEAVING_LEECH_NORMAL	0.00	prob. to leave of an incomplete non-freerider
PROB_LEAVING_SEED_FRIDER	0.90	prob. to leave of a complete freerider
PROB_LEAVING_LEECH_FRIDER	0.10	prob. to leave of an incomplete freerider

Table 8.3: Enter/Leaving control parameters.
(the listed values are related to the experiments described in §9.5.1)

perform its regular initialisation (e.g. performing a massive latency measurement).

- (3) *Entering phase*: every certain ticks (ENTERING_PEER_INTERVAL), a percentage of agents (NUM_ENTERING_PEERS over MAX_PEERS) may enter the system with a given probability (PROB_ENTERING_PEERS). The entering agents will have a certain probability (PROB_ENTERING_COMPLETED) of having the datum and a random bandwidth and cluster among the ones suggested by the reference network topology.

- (4) *Involuntary leaving phase*: every certain ticks (PEER_FAILURE_INTERVAL), a percentage of agents (POSSIBLE_NUM_PEER_FAILURES over MAX_PEERS) may involuntarily leave the system with a given probability (PROB_PEER_FAILURE).

- (5) *Voluntary leaving phase*: every certain ticks (LEAVING_PEER_INTERVAL), any agent may leave the system depending on its profile: it can be *normal* or *freerider*. A normal agent has a very low probability to leave the system when it does not have the datum (PROB_LEAVING_LEECH_NORMAL) and a low probability to do it when it has the datum (PROB_LEAVING_SEED_NORMAL) —i.e.

it contributes to the sharing process. In contrast, a freerider agent has a low probability to leave the system when it does not have the datum (PROB_LEAVING_LEECH_FRIDER) and a high probability to do it when it has the datum (PROB_LEAVING_SEED_FRIDER).

- (6) *Final inactive phase*: finally, when only a few agents lack the datum (MIN_LEECHES_TO_STOP_EL) the entering/leaving system becomes inactive.

In any case, the model does not allow agents to enter when there is already the maximum allowed population (MAX_PEERS) nor to leave when there is already the minimum allowed population (MIN_PEERS_CONNECTED). So, these parameters help to keep the agent population size within a desired range. Even more, the model does not allow an agent to leave the system if it is the only one that has the datum. This way, the sharing network is always active.

8.4.1.2 Implementation

Implementing the described entering/leaving extension model involves updating several P2PMASsim components, as highlighted in figure 8.15. In short, there is a new subcomponent (`EnterLeave`), within the Setup component (`setup`), which keeps all the entering/leaving model parameters described in previous subsection. It is registered into the Repast scheduler in order to be called at the appropriate intervals in order to perform the *Entering phase* (number 3 in §8.4.1.1) and the *Involuntary leaving phase* (4). In case there should be an entering agent –according to some random numbers and model parameters– it creates a new instance of the corresponding agent (`twoLama.dl.Peer` or `bitTorrent.Peer`). Analogously, if there should be an involuntary leaving agent, it sets the corresponding agent as offline. In addition, the agent base implementation (`agentLayer.AgentBase`) was updated to register into the scheduler in order to perform the *Voluntary leaving phase* (5) depending on the model parameters.

However, these changes entail updating other components. For example, if an agent can leave at any moment, then the messages towards it that are travelling along the network cannot be delivered. This implies updating the network infrastructure (`netLayer.infrastructure`) in order to let its components (`Component`) detect such situations and send back a new protocol message to the sender (`MsgType`, new message `err_no_dst`) to denote that the recipient agent was offline. Besides, the network adapter (`Adapter`) offers an offline switch, which stops sending packages when and agent is disconnected.

In addition, the entering/leaving *Initial phase* (1) is performed by the creation method of the context component (`setup.Context`) by using the mentioned new subcomponent (`EnterLeave`). This context component creates the initial agents depending on the reference network topology. Hence, the network component (`netLayer.Network`) has two topologies, the reference one and the actual one. Moreover, the topology component (`Topology`) supports adding and removing elements, thanks to some extensions of the underlying graph component (`tools.datastructures.Graph`).

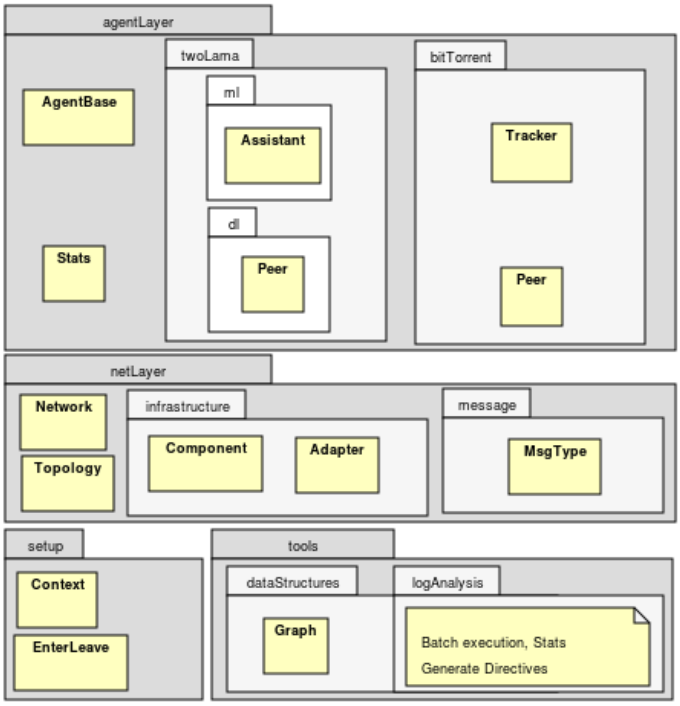


Figure 8.15: Simulator's Entering/Leaving extension components.

All described changes involve also updating the protocol, and consequently agents code (`Assistant`, `Tracker` and `Peers`). For instance, now an assistant can receive join messages at any moment. This means that its internal state machine is updated to handle this situation and ask a newcomer to measure its latency to the rest of agents in its cluster. In this case, the agent (`Peer`) will measure its latency using a shorter message than before –see §6.7.2– because there is already network traffic. The resulting measures make the assistant recompute the proper social structure and send the corresponding contact messages to agents.

Furthermore, the protocol has new messages to inform that an agent is leaving (`MsgType`, new messages `leave_p2p`, `leave_p2a`, `leave_a2a`). In particular, when an agent is leaving, it notifies its contacts (messages `leave_p2p`) and assistant (message `leave_p2a`). Then, the assistant notifies the other assistants about such fact (message `leave_a2a`). This way, domain-level agents can update their internal contact lists –and change its source when necessary– and assistants can recompute the best social structure. In an open MAS context, a leaving agent may not notify others anyway. Is such a case, other agents usually deduce that it is leaving when there are network timeouts. In our implementation, this happens when an agent sends a message to another one and receives the previously introduced `err_no_dst` message.

```
FORCE_INITIAL_PEERS = [
[ id:'P01', datum:false, frider:true,  bw:40, cluster:1 ],
[ id:'P02', datum:false, frider:true,  bw:40, cluster:1 ],
[ id:'P03', datum:false, frider:true,  bw:60, cluster:1 ],
[ id:'P04', datum:false, frider:true,  bw:80, cluster:1 ],
[ id:'P05', datum:false, frider:false, bw:40, cluster:2 ],
[ id:'P06', datum:false, frider:false, bw:40, cluster:2 ],
[ id:'P07', datum:true , frider:true,  bw:90, cluster:2 ],
[ id:'P08', datum:false, frider:true,  bw:90, cluster:2 ],
[ id:'P09', datum:false, frider:true,  bw:30, cluster:3 ],
[ id:'P10', datum:false, frider:true,  bw:50, cluster:3 ],
[ id:'P11', datum:false, frider:true,  bw:60, cluster:3 ],
[ id:'P12', datum:false, frider:false, bw:90, cluster:3 ]
]
FORCE_ENTER_LEAVE = [
75:  [ 'enter': [ [ id:'P13', datum:false, frider:false, bw:40, cluster:1 ] ] ],
275: [ 'voluL': [ [ id:'P04', datum:false ],
                  [ id:'P05', datum:false ] ],
     ],
375: [ 'voluL': [ [ id:'P01', datum:true ], ],
       'enter': [ [ id:'P14', datum:false, frider:false, bw:60, cluster:3 ], ],
     ],
475: [ 'failL': [ 'P10', 'P14' ] ]
]
```

Figure 8.16: Entering/Leaving directive example.

Finally, these new behaviour implies new sort of statistics, like the number of peers that actually entered/left the system or the number of complete datum transmissions —this number can be greater than the number of agents at the end of execution, since some completed agents may have left before. These new metrics are collected by an updated statistics component (`agentLayer.Stats`) and processed by an updated analysis stack (`tools.logAnalysis`). In fact, the batch execution was updated in order to save the entering/leaving activity during an execution —we call it an *Entering/leaving directive*. This way, in a subsequent execution, instead of rolling dices, the described methods may mimic the stored directive. Figure 8.16 shows an example of such directive. It contains the list of initially connected peer agents (`FORCE_INITIAL_PEERS`) with information about its datum possession (`completed`), its entering/leaving profile (`frider`, which stands for freerider), its individual bandwidth (`bw`) and the cluster where they are connected (`cluster`). Also, it contains a list of entering/leaving events (`FORCE_ENTER_LEAVE`). This list is indexed by the simulation tick, and the corresponding entries may have entering (`enter`), voluntary leaving (`voluL`) and involuntary leaving (`failL`) directives.

8.4.2 Norm violations

In addition to entering/leaving agents, another characteristic of open MAS is dealing with non-compliant agent populations. In particular, we focus on populations where there is a certain percentage of agents that violate norms —so-

called *violators*. Moreover, we consider that agents may have different ways of violating such norms. For instance, in the traffic scenario when the speed limit is 100 km/h (kilometres per hour), 30% of agents may violate such limit in different proportions. For example, a moderate violator may drive at 130 km/h, whereas an extreme violator may drive at 200 km/h. Actual violations also depend on other factors like system status, agent capabilities, or environmental characteristics. For instance, the mentioned extreme violator may wish to drive at 200 Km/h, but current traffic flow may actually prevent it from driving at more than 160 Km/h.

Accordingly, the norm violations extension offers different parameters to control the percentage of violators and their degree of violation. Furthermore, current P2PMASsim also offers a reward/sanction extension that goes beyond the scope of this thesis, see [Ávila et al., 2011].

8.4.2.1 Model

The norm violations model provides two parameters to describe an agent population:

- the *percentage of violators* ($pVio$): it defines the amount of agents that violate norms as the proportion of the agent population that does not fulfil norms (violators). For example, when there are 12 peers and $pVio = 0.3$, there are 30% of violators that result in 4 non-compliant peers ($12 \cdot 0.3 \approx 4$).

- the *degree of violation* ($dVio$): it establishes the extent in which an agent violates the norm. For example, if current $max_{BW} = 50\%$ and $dVio = 0.3$, then a violator may use 65% of its nominal BW ($50 + 50 \cdot 0.3 = 65$). In contrast, if $dVio = 0.7$ a violator may use 85% of its nominal BW ($50 + 50 \cdot 0.7 = 85$).

Equation 8.1 shows how violator agents compute their internal limits ($vioMax_{FR}$, $vioMax_{BW}$) according to current normative ones (max_{FR}, max_{BW}) and their degree of violation ($dVio$). Notice that the actual violation also depends on other factors like system status, agent capabilities, or environmental characteristics. For instance, a violator would not be sending the data to more agents than those that requested it, or it can not use more than 100% of its bandwidth.

$$\begin{aligned} vioMax_{FR} &= max_{FR} + max_{FR} \cdot dVio \\ vioMax_{BW} &= max_{BW} + max_{BW} \cdot dVio \end{aligned} \tag{8.1}$$

8.4.2.2 Implementation

Previous Norm Violation model mainly involves the initial population creation and the agent implementation. Specifically, as 2-LAMA approach is the only implemented coordination model that has norms, its domain-level agents and context initialisation are the components updated as depicted in figure 8.17.

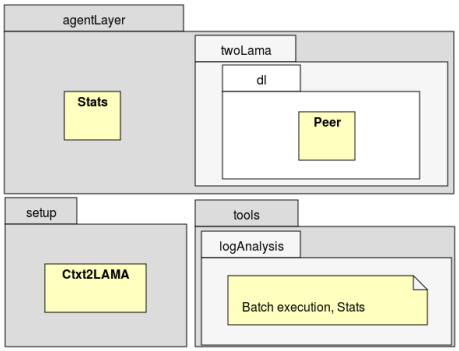


Figure 8.17: Simulator's Norm violations extension components.

On the one hand, the first Norm Violation model parameter ($pVio$) is used by the 2-LAMA context (`Ctxt2LAMA`) when creating the agent population. In particular, every time it creates a new domain-level agent, it creates a violator or a non-violator agent depending on a random dice based on the $pVio$ probability. Alternatively, 2-LAMA context can create certain agents as violators or non-violators depending on a *Norm Violations directive* —i.e. a list of violators/non-violators agent identifiers. This way, other simulations can be conducted with the same agent population in order to do a fair comparison of their results — see §9.5.2.1.

On the other hand, the domain-level agent (`twoLama.dl.Peer`) is updated to be able to violate norms in case it was created as a violator agent. Particularly, if it is a violator, it computes its internal limits as defined in eq. 8.1 —instead of just fulfilling current norms. In fact, it recomputes these limits every time that the norms are updated. This way, it sets its own internal bandwidth limit ($vioMax_{BW}$) on its network adapter –see `Adapter.relativeUpBW` in §8.3.5.2– which may be larger than the allowed by current max_{BW}. Also, it may serve the datum to a new agent if it is already serving current max_{FR} agents but less than its internal limit ($vioMax_{FR}$).

Besides, this new behaviour implies new sort of statistics, like the average of the number of datum transmissions that violate $normFR$ or the average of used bandwidth that violates $normBW$. These new metrics are collected by an updated statistics component (`agentLayer.Stats`) and processed by an updated analysis stack (`tools.logAnalysis`).

Chapter 9

Experiments

This chapter presents the experiments conducted to empirically evaluate the 2-LAMA in the P2P scenario. These experiments compare the different coordination models –presented in previous chapters– using the developed simulator. Such coordination models include the standard BitTorrent protocol as a baseline and different 2-LAMA approaches. In the experiments, the metric used to evaluate the performance is the total sharing time to spread the data among all agents, nevertheless some other metrics are also analysed. The derived results show that our proposal is able to improve system performance, specially the approaches that adapt norms. Additionally, we include a preliminary analysis about 2-LAMA robustness when facing some open MAS issues: entering/leaving agents and norm violations. The corresponding tests arise some positive preliminary results and constitute a guidance to design further experiments in such a context.

9.1 Introduction

Previous chapter 8 described the simulator we developed in order to empirically test our 2-LAMA proposal in the P2P sharing network scenario —see the corresponding specification in chapters 6-7. Specifically, the simulator P2PMASsim offers the implementation of all the suggested 2-LAMA alternatives which differ in which organisational components they adapt and how they do it —see §7.2, §7.4 and §7.5. Moreover, P2PMASsim also has an implementation of the baseline standard BitTorrent protocol —see §6.2. Therefore, the following empirical tests are able to compare any of these coordination models.

Experiments analyse several metrics, being the time to spread the data among all agents the one used to evaluate the performance. In fact, this metric is the most relevant in real P2P sharing networks. The other metrics are mainly used to roughly overview what is conditioning the resulting times. Moreover, we have frequently analysed some particular simulation logs in order to determine what is exactly happening. Hence, our comments about system behaviour are

based on all these indicators.

All in all, we present the measured metrics, an analysis about the significance of the performance results and a particular norm adaptation example to illustrate the overall behaviour. In general, these results show that our 2-LAMA proposal helps to improve system performance, specially the approach that uses machine learning to adapt norms.

Additionally, there is also a first analysis about 2-LAMA robustness when facing some open MAS issues. Designing the corresponding experiments has risen several complexities concerning how to fairly compare different approaches in this context. Even more, the number of executions to perform such experiments increases significantly. In despite of these difficulties, the corresponding tests arise some positive preliminary results and provide a guidance to design further experiments about these issues.

In summary, the empirical tests show that our proposal 2-LAMA proposal is able to help to improve system performance in the P2P sharing network scenario and that it is robust when dealing with entering/leaving participants or non-compliant agent populations.

9.2 Coordination models

The target of 2-LAMA is to adapt the coordination model of the domain-level agents in order to improve system performance. In particular, chapter 7 proposes different mechanisms to adapt different aspects of the coordination model. Hence, in order to test 2-LAMA empirically, we compare the performance of each of these adaptive coordination model alternatives. Moreover, as the 2-LAMA specification in the P2P sharing network scenario is based in the standard BitTorrent protocol, we also compare the mentioned adaptive models to the non-adaptive approach BitTorrent. This section provides further details about all these coordination models.

9.2.1 Non-adaptive coordination model

We regard a BitTorrent network (BT) as an agent-centred multi-agent system (ACMAS) approach, in which agents (peers) stablish their own organisation. Moreover, we construe it as a non-adaptive coordination model, since its agents always use the same mechanisms to coordinate. In short, its coordination model does not limit bandwidth usage and always restricts to three the number of leeches (peers lacking the datum) that a seed (peer having the datum) can start serving simultaneously.

As detailed in §6.2, we have implemented a BitTorrent version based on the original standard protocol. The main difference is that in our implementation the data have only a single piece. The rest is equivalent, so BT has a single agent (tracker) that informs about connected peers, instead of providing further assistance like our distributed meta-level. Following the original protocol, new peers without data (interested) show their interest to peers having

the datum (sources). Afterwards, sources start serving only at given intervals (`unchoke_interval`). In particular, at these intervals, a source peer communicates to three of its previously interested peers (selected peers) to say that it can serve them. Next, these peers can request the datum and all of them are served. The *selected peers* are those that were interested most recently. If two of them were interested at the same time, the one with the larger network bandwidth (`upload_bw`[1]) is selected. In fact, if a peer's interest is older than a defined interval (`ageing_period`), its age is ignored and only its peer's `upload_bw` is compared. In addition, in two out of three `unchoke_interval` selection processes, the third peer is *randomly* selected.

In our experiments, BT uses an `unchoke_interval` of 250 time units (ticks). It is approximately the time required to send three data messages in current topology —along an average peer individual link. Thus, it is the average time that a server peer can invest sending data to three selected peers —i.e. the number of simultaneous starting servings in BT. Also, we use an `ageing_period` of 130 ticks to keep the ratio defined by the official protocol.

9.2.2 Adaptive coordination models

In contrast to BitTorrent, we assume 2-LAMA has an adaptive coordination model, since its meta-level provides different mechanism to adapt the organisation of domain-level agents —see chapter 7. In particular, we present three alternatives:

- 2-LAMA with Social relationships adaptation (2L-S) but no norm adaptation.
- 2-LAMA with Social relationships adaptation and Norm adaptation using a Heuristic approach (2L-S-N-Heu).
- 2-LAMA with Social relationships adaptation and Norm adaptation using a CBR learning approach (2L-S-N-CBR).

The first alternative (2L-S) provides the social relationships adaptation detailed in §7.2. Hence, it differs from BT in their social structure. While BT has a flat social structure in which all peers contact among them, 2L-S uses our suggested two-level multi-agent architecture. In this architecture, the meta-level assists peers to create a social structure depending on their communication latencies and datum possession.

The second alternative (2L-S-N-Heu) provides also the same social relationships adaptation –since it has the 2-LAMA architecture– plus the norm adaptation detailed in §7.4. Hence, it differs from BT in their social structure but also

[1] In a multi-piece scenario, this measure is estimated from previous piece interchanges. However, since in a single-piece implementation no estimation can be performed, its actual value is taken from the network topology. In contrast, our approach (2-LAMA) is actually estimating connectivity by sending partial data messages —see §6.7.3. Notice that this gives advantage to BT as peers do not have to exchange extra data messages.

in the norms agents use to exchange data. In BT, these norms are fixed by the protocol specifications, whereas in 2L-S-N-Heu they evolve along the execution.

Analogously, the third alternative (2L-S-N-CBR) also provides the same social relationships adaptation but a different norm adaptation mechanism, the machine learning approach detailed in §7.5.

Besides, in order to make a fair comparison among BT and these 2-LAMA approaches, we use a set of initial norm parameters that provide equivalent starting conditions. Specifically, we use $\texttt{max}_{\texttt{Has}} = \infty$ –i.e. no communication restriction among clusters– $\texttt{max}_{\texttt{BW}} = \texttt{100}\%$ –i.e. no bandwidth restriction– and $\texttt{max}_{\texttt{FR}} = \texttt{3}$ —equivalent to BT serving restriction. In addition, we empirically established the adaptation interval $adapt_{interval} = 50$ time steps as large enough to let new norms influence agents' behaviour before evaluating their effects. Currently, in order to perform these adaptations, assistants aggregate their local and remote information by giving them the same importance (i.e. equal local and remote weights, $\forall_j w_{R,j} = w_L$, see §7.3.1.3). Besides, the machine learning approach (2L-S-N-CBR) requires some more parameters, like the minimum similarity threshold `MIN_SIM`=0.65, the learning rate $\alpha_g = 0.1$, the t_{spread} revision factor $k_g = 0.1$ or the attribute weights (w_i^{Θ}). We computed these attribute weights using PRS [Salamó and López-Sánchez, 2011] weighting method, and obtained: $w^{\Theta}_{Compl} = w^{\Theta}_{SrvBW} = w^{\Theta}_{RcvBW} = w^{\Theta}_{RcvEffBW} = w^{\Theta}_{Waiting} = 1$ and $w^{\Theta}_{SeedBW} = w^{\Theta}_{LeechBW} = 0.5$ —see §7.5.2.

9.3 Experiment design

All mentioned coordination models have been executed using the network topology depicted in figure 6.2b. Notice that in BT approach, a single tracker is linked to $r0$, whereas in our 2-LAMA approaches there is an assistant connected to each ISP ($r1..r3$). In any case, these elements (tracker/assistants) have an infinite bandwidth —as if they were located at the ISP. We have used data messages of 5000 data units, and the rest of messages of a single data unit. Additionally, peers exchange messages of 150 data units to estimate the communication latencies between them —see "`lat_req`"/"`lat_rpl`" messages in §6.7.2.

Overall, models have been tested by varying the domain-level agent that initially has the datum. However, depending on model's properties we performed a different set of executions. They are illustrated in figure 9.1 and detailed as follows.

On the one hand, due to the random nature of the BitTorrent (i.e. some served peers are selected haphazardly), the BT results show the average of repeating 50 times the execution with the data in each possible initial position (i.e. $12 \times 50 = 600$ executions, where the 12 corresponds to all possible initial data positions, and the 50 corresponds to repetitions).

On the other hand, the only social relationships adaptation approach and the heuristic approach simply need to be executed once on each initial position (i.e. 12 executions) since 2L-S / 2L-S-N-Heu have neither randomness –like BT– nor training phase —like 2L-S-N-CBR.

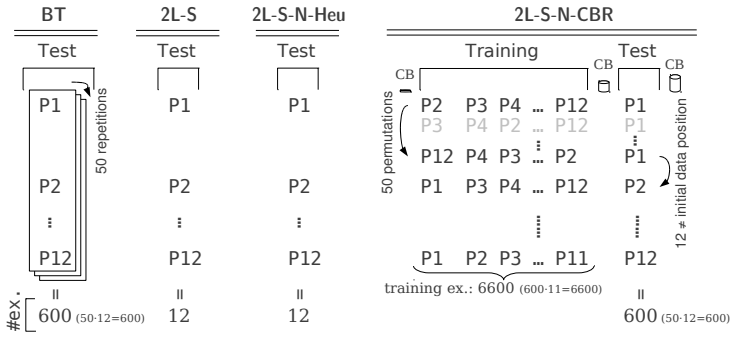


Figure 9.1: Executions performed in each coordination model.

Besides, the CBR learning approach requires a training period before testing each initial data position. Hence, the system is trained with executions where the data is in all the other possible initial positions. This lets the system learn and fill the initially empty case-base —either by calling the heuristic or adapting previously learnt cases. Once it has been trained, the system is tested against the remaining initial data position —the one not used during the training. Given that the order of data positions during the training is relevant, different permutations of initial positions should be tested. In particular we tested 50 random permutations out of all possible permutations of the 11 positions (i.e. 11!). Consequently, the results of 2L-S-N-CBR show the average of executing 50 times each test of initial position (i.e. $12 \times 50 = 600$ executions, where the 50 corresponds to different training permutations for the 12 tested initial data positions).

Accordingly, the results shown in next section correspond to the average of the total number of test executions shown in the bottom part of figure 9.1. For instance, the 2L-S-N-CBR tests imply 600 test executions and their corresponding 6600 training executions.

9.4 Results and analysis

Performance in our experiments is measured in terms of the total *time* (t_{spread}) required to spread the datum among all agents. In addition, we also measure the network metrics defined in §6.3.2: the *network cost* (c_n) consumed by all messages, the average number of links traversed by each message (*hops*), the *network usage* ($netUsg$) of all channels along a sharing process and the *network saturation* ($netSat$) which indicates if there were messages waiting to traverse those channels. In addition, we consider two more values about the case-base for the machine learning approach: $\#cases$ as the average of the total number of cases at the end of a test execution and *fromHeu* as the percentage average of cases in the case-base with a solution generated by calling the heuristic — see §7.5.3.1-7.5.3.2. Table 9.1 shows theses measures averaged over the number

Table 9.1: Results (average for all executions).

	time	*net cost* (c_n)	*hops*	*netUsg*	*netSat*
BT	986.2	206,592.0	3.42	0.098	0.177
2L-S	793.1	338,448.3	3.22	0.163	1.813
2L-S-N-Heu	744.5	306,755.0	3.03	0.146	2.126
2L-S-N-CBR	732.6	348,399.6	3.25	0.161	1.846
#cases =155.4					
fromHeu =1.3%					

of executions –see §9.3– for the different coordination models –see §9.2– in the P2P sharing network scenario. Next subsections analyse these results, showing that 2-LAMA presents a significantly good performance, specially the machine learning approach.

9.4.1 Performance evaluation

If we compare the performance of BT and 2-LAMA coordination models, we see in table 9.1 that 2-LAMA approaches require less *time* to share the datum —see next subsection 9.4.2 for further details about the significance of these result. This means that the time invested in communicating with our suggested meta-level is less than the time benefit of having this additional level. In contrast, the *network cost* is larger in 2-LAMA. This indicates that, in our approaches, the network is intensively used along the whole execution without achieving saturation —otherwise, the time would increase. Having a meta-level implies that coordination messages are exchanged among peers and assistants and also between assistants. However, the network usage increment is mainly caused by transmitting more data messages. In particular, in 2-LAMA there are more data messages because some of them are not totally transmitted when: (i) a destination peer sends a cancel message to its source because it has found a faster one or (ii) a source peer stops sending data to fulfil an updated max_{FR}. Thus, we expect to minimise the network consumption related to compare sources (i) when dealing with more than one piece of data, since peers could compare sources depending on the pieces previously retrieved. In more detail, as pointed out in §9.2.1, current BT implementation takes the `upload_bw` measure from the network topology information, whereas, in 2-LAMA implementation, peers receive data simultaneously from more than one source to compare their bandwidths —see §6.7.3. However, in a multi-piece scenario, these measures could be estimated from previous piece exchanges, so 2-LAMA would save the network cost associated to having simultaneous sources. Moreover, currently 2-LAMA invests an average of 20% of its network cost and 10% of its sharing time to perform its social structure phase —see §6.7.2. Such investment in time and network resources would be exploited longer if more pieces were exchanged. In summary, a multi-piece scenario would benefit more 2-LAMA than BT.

Regarding the number of links traversed by messages (*hops*), our 2-LAMA

approaches have more local communications –i.e. intra-clusters– than BT. This is convenient because local messages have lower latencies and costs, since they are usually performed within the same cluster. For example, the heuristic norm adaptation approach (2L-S-N-Heu) has 12% less hops in average than the baseline BT. However, in our case study this is not the only factor that contributes to have a short time, since the machine learning approach (2L-S-N-CBR) is the fastest and its hops average is larger than in the other 2-LAMA approaches.

Finally, the results in table 9.1 show that the norm adaptation approaches (2L-S-N-Heu, 2L-S-N-CBR) perform better than the 2-LAMA approach that just adapts the social structure (2L-S), since they require less time. In fact, in our tests, no single combination of norm parameters applied during a whole execution outperforms an execution that starts by using the same parameter combination but adapts it later on —see also [Campos et al., 2008]. Furthermore, our learning approach (2L-SS-N-CBR) achieves better results than our heuristic approach (2L-SS-N-Heu). This means that our heuristic performs a good estimation of the mapping between system status, norms and outcomes, but current learning approach 2L-S-N-CBR automatically improves this estimation —see next subsection 9.4.3 for an example. However, in order to obtain better results, the CBR approach requires a computation more complex than the heuristic one. Specifically, the heuristic is a set of nested conditional instructions, so its cost is $O(k)$. Whereas, the CBR approach requires computing the similarity of current problem to all previous cases, so its cost is approximately $O(n)$ —n stands for the number of previous cases. Favourably, the number of cases is not large, as shown in $\#cases = 155.41$ at table 9.1. Hence, the actual CBR's computation cost is small. Moreover, our CBR approach is basically obtaining solutions by adapting previous solved cases, since the heuristic is only called a 1.3%, see $fromHeu$ at table 9.1.

9.4.2 Significance tests

We also analyse whether there are significant differences between the presented approaches by applying the Friedman test [Friedman, 1937, Friedman, 1940] and Nemenyi test [Nemenyi, 1963] to the time results. First of all, we compute the *mean rank* (r) of each algorithm considering all the experiments ($k = 4$ algorithms and $N = 600$ different experiments[2] for each test). We rank alternative algorithms, for each experiment, following the practice of Friedman. The rankings are obtained computing each particular ranking r_i^j for each experiment i and each coordination model j, and computing the mean ranking R for each model as $R_j = \frac{1}{N}\sum_i r_i^j$, where N is the total number of experiments. Compared with mean performance values, the mean rank reduces the susceptibility to outliers which, for instance, allows a classifier's excellent performance in one query to compensate for its overall poor performance [Demšar, 2006]. Secondly, we apply the Friedman and Nemenyi tests to analyse whether the difference between

[2] In 2L-S and 2L-S-N-Heu there are 12 experiments, so their results have been repeated 50 times to be able to compare them with the rest of approaches.

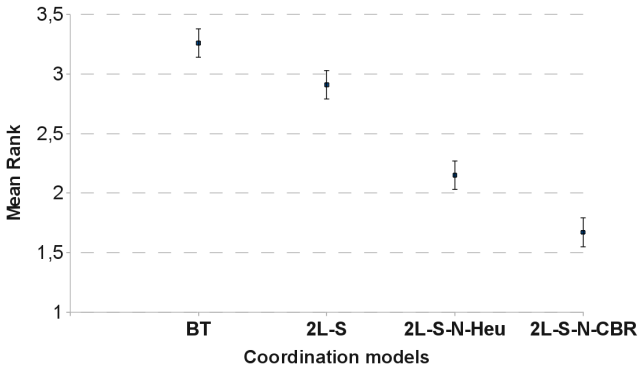


Figure 9.2: Application of the Nemenyi test to time results on the different coordination models.

algorithms is statistically significant.

The Friedman test, recommended by [Demšar, 2006], is effective for comparing multiple algorithms across multiple data sets —in our case, across multiple experiments. It compares the mean ranks of algorithms to decide whether to reject the null hypothesis, which states that all the methods are equivalent and therefore their ranks should be equal. The Friedman statistic value is computed as follows:

$$X_F^2 = \frac{12N}{k(k+1)} \left[\sum_j R_j^2 - \frac{k(k+1)^2}{4} \right] \tag{9.1}$$

where k is the number of algorithms to compare.

Since this value is undesirably conservative, Iman proposed a corrected statistic [Iman and Davenport, 1980]:

$$F_F = \frac{(N-1)X_F^2}{N(k-1) - X_F^2} \tag{9.2}$$

When we apply the Friedman test in our experimental setup with k algorithms and N different experiments, F_F is distributed according to the F distribution with $(4-1) = 3$ and $(4-1) \cdot (600-1) = 1797$ degrees of freedom. The critical value of $F(3, 1797) = 2.61$ is computed at the 0.01 critical level —i.e. at 99%. For our experiments on the P2P sharing network scenario we obtained the values of $X_F = 542.55$ and $F_F = 258.4$. As the value is higher than 2.61 we can reject the null hypothesis.

Once we have checked for the non-randomness of the results, we can perform a post hoc test to check if one of the approaches can be singled out. For this purpose, we use the Nemenyi test. In brief, it states that two approaches are significantly different if the corresponding average ranks differ by at least the critical difference value:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \tag{9.3}$$

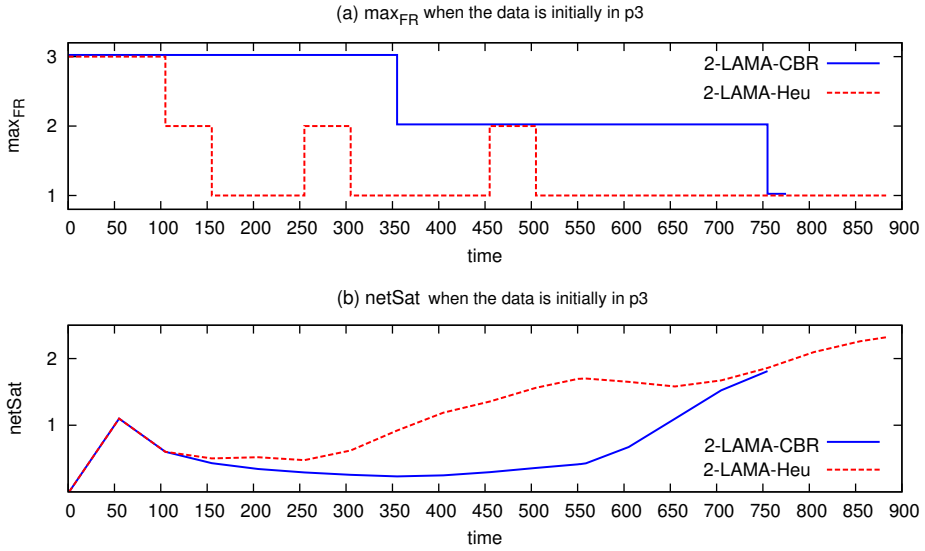


Figure 9.3: System evolution example in terms of a) norm adaptation and b) resulting saturation.

where q_α is based on the Studentized range statistic divided by $\sqrt{2}$ and k is the number of algorithms to compare. In our case, when comparing k algorithms with a critical value $\alpha = 0.01$, $q_{0.01} = 3.113$ for a two-tailed Nemenyi test —i.e. 99%. Hence, the critical difference value is $CD = 0.232$. Thus, for any two pairs of algorithms whose rank difference is higher than CD, we can infer –with a confidence of 99%– that there exists a significant difference between them.

Figure 9.2 illustrates the results of the Nemenyi test. In the figure, dots represent the mean rank of each coordination model. Vertical lines across dots indicate the critical difference (CD). As stated, the performance of two approaches is significantly different if their corresponding mean ranks differ by at least the critical difference. For instance, figure 9.2 reveals that 2L-S is significantly better than BT. Overall, the best mean rank corresponds to 2L-S-N-CBR which also shows that is significantly better than the rest of coordination models at a 99%.

9.4.3 Norm adaptation example

In order to illustrate how the machine learning approach (2L-S-N-CBR) improves the heuristic approach (2L-S-N-Heu), figure 9.3.a shows the adaptation of $normFR$ in a single execution –where the datum is initially at peer p3– using both approaches. In brief, the heuristic tends to force a small $\texttt{max}_{\texttt{FR}}$ to avoid network saturation but CBR learns that it can obtain better results (775 ticks versus 883 ticks) by using a larger value. Furthermore, the network saturation curves shown in figure 9.3.b reveal that the heuristic has actually more network

saturation (*netSat*) when it starts using a smaller max_{FR} (i.e. $t = 100$). In fact, inspecting execution details it arises that lowering max_{FR} values does not always imply a saturation reduction. For instance, this situation can be found when a source agent (seed) with a large bandwidth is serving data to other agents (leeches) with smaller bandwidths. In such a case, it may be worse to serve just a few agents, since there may be more probability to saturate their individual links. Avoiding this saturation, the CBR approach is able to use the network more intensively and thus, it obtains a shorter sharing time. Regarding figure 9.3.b, its initial peak reflects intensive communication used to estimate bandwidths among peers. Next, the CBR approach has a low net saturation (if $netSat < 1$ then, in average, links have less information to transmit than their nominal bandwidth) since different seeds send data to different leeches in parallel. However, at the end the of the execution, the *netSat* increases because the links of the few remaining leeches can be saturated by some of the numerous available seeds.

9.5 Exploring open MAS issues

In addition to the results presented in previous sections, we performed some exploratory experiments about the open MAS extensions discussed in §8.4. Conceiving such experiments has risen several design issues about how to fairly compare different approaches in an open MAS context. Even more, it has risen that the number of executions to perform such experiments may increase significantly.

In particular, this section presents an initial exploration of two specific open MAS issues: MAS where there are agents entering and/or leaving during the execution (Entering / Leaving agents tests) and MAS where there are non-compliant agents (Norm violations tests). In each case, there is a subsection detailing a test design proposal to fairly compare different approaches on such context. These proposals may be a guidance for further tests on different scenarios and approaches. Besides, there is also a subsection containing some preliminary results on these contexts. These results are preliminary because due to time restrictions, we could only explore a small subset of the possible number of executions in order to keep an affordable number of simulations.

Overall, these experiments show that 2-LAMA's architecture can deal with such open MAS contexts and the preliminary results suggest that current implementation is robust enough in such contexts.

9.5.1 Entering / Leaving agents

One characteristic of open MAS is that it is not feasible to control when agents leave the system and it is usually worth to let them enter along the execution. For instance, in real P2P sharing networks, connected computers may leave the system due to a net failure or as soon as they get the data —instead of sharing it for a while. In any case, such changes in agent population may alter domain-level activity. Moreover, in 2-LAMA, the meta-level may also update the coordination

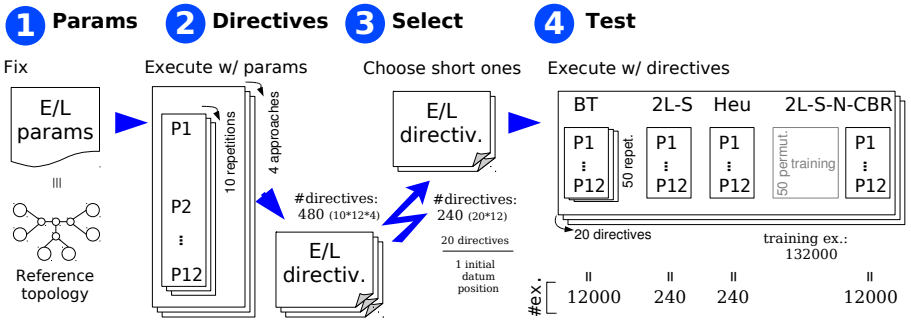


Figure 9.4: Entering/Leaving directive generation and tests.

model in order to assist the domain-level in facing such population changes. This subsection suggests how to test the robustness of different approaches when there are this sort of agent population changes.

Before dealing with entering/leaving issues, in [Campos et al., 2010] we analysed the behaviour of the heuristic approach (2L-S-N-Heu) when running different communication network topologies —i.e. different sets of agents with different communication capacities. Such topologies differ in: layout (ring/star), size (number of agents), heterogeneity (agents per cluster or agents' bandwidth). The corresponding results showed that 2-LAMA is robust when changing underlying communication network's topology. For instance, the times obtained when increasing the number of agents –in similar topologies– were sub-linear, so 2-LAMA is scalable when the population increases.

However, in that experiment, the topologies were fixed along the whole sharing process and only differ among different tests. In contrast, in current experiments we explore the behaviour of the different approaches when the topology evolves because there are agents entering/leaving along the execution. The resulting preliminary results show that 2-LAMA is robust in such context.

9.5.1.1 Experiment design

Designing an experiment to compare different MAS executions when there are agents entering or leaving the system arises some fairness issues. Such issues are related to guarantee an equivalent agent population behaviour in compared simulations. In this sense, fixing the same entering/leaving model parameters –see §8.4.1.1– and running many experiment would guarantee such equivalence. However, in order to keep an affordable number of simulations, we decided to generate some sets of entering/leaving directives –see §8.4.1.2– and compare the approaches when they follow the same directive. Figure 9.4 illustrates this process, that is detailed as follows.

First (fig.9.4-1), we fixed the entering/leaving model parameters as shown in table 8.3. Such values try to simulate a network similar to the one used in previous tests —see §9.3. For instance, given the `MAX_PEERS`=15 and

PEERS_INITIALLY_CONNECTED=0.8, the initial number of connected agents matches previous experiments topology (i.e. $15 \cdot 0.8 = 12$ agents). Then, at fixed intervals (ENTERING_PEER_INTERVAL, PEER_FAILURE_INTERVAL and LEAVING_PEER_INTERVAL) there may be agents entering or leaving, but the difference of population size will not be larger than 25% of the average population of 12 agents (i.e. $9 \leq \#peers \leq 15$, since MIN_PEERS_CONNECTED=0.6 so $15 \cdot 0.6 = 9$).

Next (fig.9.4-2), we performed 10 executions for each initial datum position –i.e. they start with the same topology than previous tests, see fig. 6.2b– using each coordination approach with the same parameter values. During these executions, we recorded their entering/leaving activity thanks to this P2PMASsim capability —see figure 8.16. As a result, we had 40 entering/leaving directives ($10 \cdot 4 = 40$, where 4 stands for the number of coordination models) for each of the 12 initial datum positions. Within this possible 40 entering/leaving directives, we chose (fig.9.4-3) the 20 shorter executions. This way, we could guarantee that all approaches could follow the whole directive, since they are not going to finish before.

Finally (fig.9.4-4), we executed the reference tests described in figure 9.1, but repeating them 20 times —i.e. once per entering/leaving directive. Accordingly, the results shown in next subsection correspond to the average of the total number of test executions shown in the bottom part of figure 9.4-4. For instance, the 2L-S-N-CBR tests imply 12000 test executions and their corresponding 132000 training executions.

9.5.1.2 Results

The results of the entering/leaving tests described previous section appear in table 9.2. It contains the average of the metrics defined in §6.3.2: the *time* (t_{spread}) required to spread the datum among all agents, the *network cost* (c_n) consumed by all messages, the average number of links traversed by each message (*hops*), the *network usage* ($netUsg$) of all channels along a sharing process and the *network saturation* ($netSat$) which indicates if there were messages waiting to traverse those channels. These metrics are available for all the coordination models described in §9.2: the base-line coordination model (BT), the *only-social-adaptation* 2-LAMA approach (2L-S), the *heuristic* norm-adaptation approach (2L-S-N-Heu) and the *machine learning* norm-adaptation approach (2L-S-N-CBR).

Notice that in terms of performance –i.e. time– all 2-LAMA approaches have still a better performance than the base-line BitTorrent (BT). Furthermore, within the 2-LAMA proposals, the machine learning approach (2L-S-N-CBR) is still faster than the rest. Observe that the number of cases ($\#cases$) has increased (from 155.41 to 168.5). This reflects that the topology changes along the execution poses more heterogeneous situations to CBR, which finds less similar previous cases and creates more cases. However, these new cases are mainly created by reusing previous ones instead of interrogating the heuristic, as shown by the decrease in $fromHeu$ (from 1.3% to 1.2%). The rest of metrics basically keep the same behaviour than in previous simulations without entering/leaving

Table 9.2: Entering/Leaving results (average for all executions).

	time	*net cost* (c_n)	*hops*	*netUsg*	*netSat*
BT	1220.69	233,810.03	3.33	0.076	0.153
2L-S	847.86	340,902.30	3.18	0.124	1.443
2L-S-N-Heu	822.80	336,891.31	3.10	0.126	1.495
2L-S-N-CBR	805.48	331,909.21	3.14	0.127	1.421
#cases =168.5					
fromHeu =1.2%					

agents —see table 9.1.

Above all, these preliminary results show that 2-LAMA is robust when there are agents entering and leaving in the P2P sharing network scenario. In order to continue this exploration, we should test more entering/leaving directives, but also more sets of entering/leaving parameters. However, it should be taken into account that any of these expansions increases the number of simulations in some orders of magnitude.

9.5.2 Norm violations

In addition to entering/leaving agents, another characteristic of open MAS is that they may be populated with non-compliant agents since the system designer cannot control their development. For instance, in a real traffic scenario, some cars may violate traffic rules.This subsection suggests how to test the robustness of different approaches when there are non-compliant agent populations.

The presented experiment focus on the machine learning approach (2L-S-N-CBR), since it has the best performance in previous experiments — the heuristic approach (2L-S-N-Heu) was already explored in a norm-violations context in [Ávila et al., 2011]. In particular, this experiment explores how the system behaves when there are different percentages of peers that excess the limits established by norms in a certain degree. We could also add sanction/reward mechanisms in such context, but [Ávila et al., 2011] concluded that they added more costs than benefits to our case study. All in all, the huge number of simulations required by these sort of tests and our time restrictions led us to focus on what was most relevant to test —it is worth to mention that current experiment has four times the number of simulations in [Ávila et al., 2011], among others, due to the CBR training phase.

Results show that the norm-adaptation coordination model is better than the no-norm-adaptation ones. In other words, 2-LAMA approach is robust enough when dealing with non-compliant agent populations.

9.5.2.1 Experiment design

The model to simulate non-compliant agent populations is detailed in §8.4.2.1. In brief, it has two parameters: the percentage of violators ($pVio$) and the degree

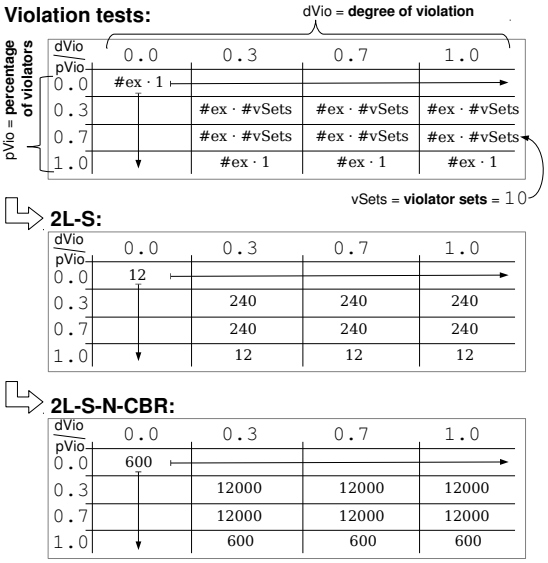


Figure 9.5: Number of executions in robustness tests.

of violation ($dVio$). The former $pVio$ defines the amount of agents that violate norms as the proportion of the agent population that does not fulfil norms —e.g. $pVio = 0.7$ means that there is a 70% of *violator* agents. And the latter $dVio$ establishes the extent in which an agent violates —e.g. $dVio = 0.3$ means that a violator may use 65% of its nominal BW when the current limit is $\texttt{max}_{\texttt{BW}} = 50\%$.

However, in order to keep an affordable number of simulations, we set both violation parameters ($pVio$, $dVio$) to a discrete set of values: 0 for no violations at all, 1 for the maximum level of violations and $\{0.3, 0.7\}$ for two intermediate violation levels. This approach is depicted in the top part of figure 9.5. Notice, that when there are no violations (either because $pVio = 0$ or $dVio = 0$), the *total number of executions* is the $\#ex$ described in figure 9.1. Similarly, when all peers are violators ($pVio = 1$), the total number of executions is also the mentioned $\#ex$ for every different degree of violation ($dVio = \{0, 0.3, 0.7, 1\}$). However, when there is an intermediate number of violators ($0 < pVio < 1$) there are several possible distinct *violator sets* ($vSets$). For instance, when $pVio = 0.3$, there are $\binom{12}{4}$ different sets of 4 ($12 \cdot 0.3 \approx 4$) violator peers out of 12. Accordingly, when some peers violate norms ($0 < pVio < 1$ and $dVio > 0$) the total number of executions is the mentioned $\#ex$ times $vSets$. Again, in order to keep an affordable number of simulations, current results take into account ten possible violator sets ($vSets = 20$).

According to the described tests, their results correspond to the total number of executions shown in the middle and bottom parts of figure 9.5. For instance, the 2L-S-N-CBR tests imply 74,400 test executions ($74,400 = 600{\cdot}4{+}12,000{\cdot}6$) and their corresponding 818,400 training executions ($818,400 =$

2L-S

pVio \ dVio	0.0	0.3	0.7	1.0
0.0	793.08	793.08	793.08	793.08
0.3	793.08	776,04	803,75	837,96
0.7	793.08	759,81	839,37	893,38
1.0	793.08	746.42	868.08	980.33

2L-S-N-CBR

pVio \ dVio	0.0	0.3	0.7	1.0
0.0	732.59	732.59	732.59	732.59
0.3	732.59	763,56	769,54	768,43
0.7	732.59	782,00	791,68	798,34
1.0	732.59	847.42	835.90	789.65

Table 9.3: Violation tests numeric results (average time for all executions).

$74,400 \cdot 11$, see also figure 9.1). All these executions are performed using the same topology than in previous tests —see §9.3.

9.5.2.2 Results

The results of the violation tests described in previous section appear in table 9.3. It contains the average of the sharing time for all described executions depending on both violation model parameters: *percentage of violators* ($pVio$) and *degree of violation* ($dVio$). The higher they are, the more violations there are during the execution. Figure 9.6 illustrates these results. Its left part shows the results for the *only social structure adaptation* 2-LAMA approach (2L-S) whereas its right part shows the results for the *machine learning norm-adaptation* approach (2L-S-N-CBR). Each part has a set of bars for each percentage of violators ($pVio$), which contains a bar for each degree of violation ($dVio$). Moreover, the base-line coordination model (BT) is depicted as a constant line that corresponds to the 986.2 ticks —see table 9.1.

Notice that, in general, the performance of the norm adaptive approach is still better than the other approaches in presence of violators. As expected, the sharing time increases when the violation rate grows —e.g. see how 2L-S-N-CBR bars clearly increase from $pVio = 0.3$ to $pVio = 0.7$ and also when $dVio$ rises. It is worth mentioning that, in general, the norm-adaptation approach presents a time shorter than 2L-S —i.e. its bars are shorter. Specially when all agents always violate norms ($pVio = 1$ and $dVio = 1$), the norm-adaptation approach is able to counter-act the violation effects since it can toughen up the limits —by decreasing their values. Furthermore, 2L-S-N-CBR presents a more steady time variation in comparison to the no-norm-adaptation approach.

In summary, results show that the norm-adaptation coordination model is better than the no-norm-adaptation ones, even when agents violate norms in the P2P sharing network scenario. That is to say, our norm-adaptation approach

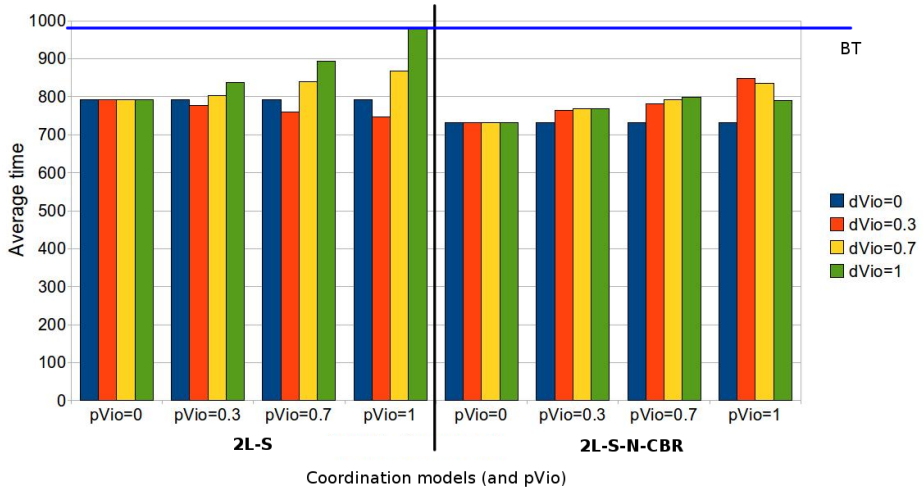


Figure 9.6: Violation tests graph results.

is robust enough in presence of non-compliant agent populations. In order to continue this exploration, there should be more violator sets ($vSets$), but also more values of both violation parameters ($pVio$, $dVio$). Nonetheless, notice that current violation experiments imply almost one million simulations and that any of the mentioned expansions would increase this number in some orders of magnitude.

Chapter 10

Conclusions

This chapter summarises the achievements of our research according to the original thesis objectives laid out on first chapter. It reviews the actual thesis development of those objectives, detailing the contributions and the derived publications. Moreover, it also suggests different paths to extend this research in the future work.

10.1 Achieved objectives and contributions

Over the years, software systems are applied to more complex and dynamic domains. Among these systems, the Organisation Centred Multiagent Systems (OCMAS) have proven to be successful in promoting a coordination model among agent that is convenient to face such complexity. Furthermore, one of the main challenges in software engineering is to design and to develop mechanisms to endow such systems with self-adaptation capabilities to continue being effective under changing situations. Our research addresses this issue when dealing with regulation-oriented problems —i.e. problems where it is not feasible to assign subtasks to agents. The result is an abstract OCMAS architecture proposal, that is able to assist agents to improve system performance on such problems: 2-LAMA.

In order to reach this general purpose, we introduced five particular objectives: (1) to define the concept of coordination support related to sustain a coordination model among MAS participants; (2) to review the previous work from this perspective in order to identify some areas that could be enhanced; (3) to propose an approach to provide assistance services to agents; (4) to consider different alternatives to provide such assistance in the form of organisational adaptation; and (5) to evaluate our approach in a P2P sharing network scenario. All in all, in first chapter we introduced these objectives, specified the target regulation-oriented problems and described the scenarios that were later used to illustrate subsequent concepts.

We achieved the first two objectives by defining our *coordination support*

perspective of MAS approaches (1) and *reviewing the state of the art* from this angle (2) —see ch. 2. This perspective is well-argued and presented in form of different service layers illustrated by several previous works. As a result, we identified a set of assistance services as a potential research area. Next, we proposed the Situated Autonomic Electronic Institutions (SAEI), which were focused on the organisational adaptation of an existing MAS by means of an overlapped institution —see ch. 3. Such approach was the antecedent of our OCMAS abstract architecture proposal. Afterwards, we schematised different assistance services as a coordination support layer that we named *Assistance Layer* —see ch. 4. This contributed to first objective (1) and provided an incipient formalisation of the this layer towards the third objective (3), i.e. proposing an approach to provide such services. This formalisation was illustrated in one of the presented scenarios, the auction house.

In particular, we focused on a single assistance service, the *Organisational Adaptation*, and we proposed an approach to provide it —see ch. 5. In fact, we proposed two alternatives: an extension to use Electronic Institutions to adapt an existing MAS (SAEI) and a more generic architecture based in a meta-level approach (2-LAMA). The second one, the *Two Level Assisted MAS Architecture*, is our actual contribution to achieve the third objective (3). In this distributed abstract architecture, the MAS organisation is updated by a set of agents that reason at a higher level of abstraction and are not involved in the domain activity —i.e. the meta-level. Such mechanism *adapts the coordination model* (organisation) that structures participants interactions whenever there are changes –in agents' behaviour or in the environment– that lead to a poor fulfilment of the system goals. Therefore, we advocate for endowing the organisation with adaptation capabilities, instead of expecting agents to be capable of adapting the organisation by themselves. Notice that we made a relevant effort to formalise the whole 2-LAMA model and its subsequent specification. This may help to apply the same model to more scenarios. For instance, during the formalisation, we provided several specification examples in the traffic scenario.

Even more, we later used our *P2P sharing network* case study to fully specify all formalised 2-LAMA components —see ch. 6. Therefore, we could elaborate different adaptation alternatives in this scenario –see ch. 7– as demanded by the fourth objective (4). In particular, we proposed how two adapt two organisational components in the P2P scenario: the net of *relationships* among agents and the *norms* that regulate participants behaviours.

Furthermore, we presented two alternatives to perform the norm adaptation. One alternative is based on a *heuristic* coded at design-time, whereas the other one is based on the *CBR* machine learning technique that evolves at runtime. The former depends totally on designer's previous knowledge and provides an adaptation fixed at design-time. This implies that the developer should think about all possible situations, otherwise the technique may be ineffective if the circumstances significantly change. In contrast, the CBR approach can use its own experience to learn how to act and it can update its behaviour under changing circumstances. In fact, we have tailored the classical CBR in order to take

into account both some heuristic knowledge and its own experience.

In relation to the last objective (5), we implemented a simulator and we used it to perform an empirical evaluation of our 2-LAMA proposal. Regarding the implementation, we developed a P2P sharing network MAS *Simulator* (P2PMASsim) —see ch. 8. This simulator provides several facilities to analyse system behaviour and compare different MAS approaches with the same initial conditions. We paid special attention to the flexibility of its internal architecture, which facilitates replacing some of its components in order to simulate other approaches and/or scenarios.

Finally, we empirically evaluated our different 2-LAMA approaches and used the standard BitTorrent protocol as a base-line —see ch.9. The corresponding results show that our 2-LAMA proposal is able to improve system performance, specially the norm-adaptation approaches. Additionally, there is also a exploratory analysis about 2-LAMA robustness when facing some open MAS issues: entering/leaving agents and non-compliant participants. This analysis provides positive results and constitutes a guidance to design further experiments on current case study or on different scenarios and/or approaches.

10.2 Publications

This section contains the list of the main publications we have done during our research work, mentioning their contributions to this thesis, the related objectives and the chapters more influenced by their contents. Notice that the list contains some journal papers, some relevant congress papers and an awarded demo, among others. These relevant publications are the following:

- Jordi Campos, Maite Lopez-Sanchez, J.A. Rodriguez-Aguilar, and Marc Esteva "*Formalising Situatedness and Adaptation in Electronic Institutions*" in Coordination, Organizations, Institutions, and Norms in Agent Systems IV . Lecture Notes in Artificial Intelligence (LNAI 5428, ISBN 978-3-642-00442-1, ISSN 0302-9743). Springer. pp. 126-139. 2009. From COIN'08 at AAMAS'08.

  - It defined our first self-adaptive OCMAS approach (SAEI) in the traffic scenario [Objective: 4; Chapters: 1,3]

- Jordi Campos, Maite Lopez-Sanchez, and Marc Esteva. "*Multi-Agent System adaptation in a Peer-to-Peer scenario*". In ACM SAC Symposium on Applied Computing - Agreement Technologies Track. Hawaii, USA, March 9 - 12, 2009 (ISBN: 978-1-60558-166-8). pp. 735-739. 2009

  - It defined our second self-adaptive OCMAS approach (2-LAMA) in the P2P sharing network scenario and concluded that the cost of having a meta-level was lesser than the benefits it reported [Objective: 4; Chapters: 1,5]

- Jordi Campos, Maite Lopez-Sanchez, and Marc Esteva. "*Norms in 2-LAMA*" Research Report at Artificial Intelligence Research Institute (IIIA), Spanish National Research Council (CSIC), ref. number RR-IIIA-2008-05. 2008.

  - It compared the results of using 2-LAMA with different fixed norm values versus using an initial norm adaptation heuristic, concluding that the adaptive approach managed to obtain a good time in despite of initial norm values [Objective: 5; Chapter: 9]

- Jordi Campos, Maite Lopez-Sanchez, and Marc Esteva. "*Coordination Support in MAS*". In AAMAS'09 Eighth International Conference on Autonomous Agents and Multiagent Systems, (Impact factor by CiteSeer = 1.18) May 10-15, 2009, Budapest, Hungary (ISBN 978-0-9817381-5-4). pp.1301-1302. 2009

  - It defined our Coordination Support perspective illustrated in both the auction house and the P2P sharing network, and presents the Assistance services as a step forward in MAS development [Objectives: 1,2; Chapters: 1,2,4]

- Jordi Campos, Maite López-Sánchez, Marc Esteva, Alba Novo, Javier Morales "*2LAMA Architecture vs. BitTorrent Protocol in a Peer-to-Peer Scenario*" Artificial Intelligence Research and Development, proceedings of the 12th International Conference of the Catalan Association for Artificial Intelligence. Edited by Sandra Sandri, Miquel Sanchez-Marré, Ulises Cortés. Vol 202 of the series Frontiers in Artificial Intelligence and Applications 202. IOS Press. ISBN. 978-1-60750-061-2, ISSN 0922-6389. pp. 197-206, Cardona Barcelona, 21-23 Oct 2009.

  - It defined our simplified version of the BitTorrent protocol and empirically compared it to our 2-LAMA approach, concluding that our meta-level outperforms this base-line protocol [Objective: 5; Chapters: 6,9]

- Jordi Campos, Marc Esteva, Maite Lopez-Sanchez, and Javier Morales "*An Organisational Adaptation Simulator for P2P networks*". H. Aldewereld, V. Dignum and G. Picard (Eds.) Spinger. Engineering Societies in the Agents World X. Lecture Notes in Artificial Intelligence. LNAI 5881, ISSN 0302-9743, ISBN 978-3-42-10202-8, 10th International Workshop ESAW'09, pp. 240-242, Utrecht, The Netherlands, 16-19 November 2009.

  - We won the Best Demo Award by presenting the features of our simulator [Objective: 5; Chapter: 8]

- Jordi Campos, Maite Lopez-Sanchez, Marc Esteva and Javier Morales "*A simulator for organisation-centred MAS adaptation in P2P sharing networks*" proceedings of the ninth international conference on Autonomous Agents and MultiAgent Systems AAMAS'10 (Impact factor by CiteSeer = 1.18) pp. 1615-1616. Toronto, Canada, May 10-14 2010.

  - It demoed P2PMASsim at the main conference, with an enhanced control through the GUI [Objective: 5; Chapter: 8]

- Jordi Campos, Maite Lopez-Sanchez and Marc Esteva "*Norm Adaptation using a Two-Level Multi-Agent System Architecture in a Peer-to-Peer Scenario*" proceedings of the ninth workshop on Coordination, Organization, Institutions and Norms in multi-agent systems (COIN) at AAMAS'10 pp. 64-71. Toronto, Canada, May 11 2010.

  - It detailed current heuristic approach to adapt norms in 2-LAMA, concluding it outperformed the no-norm-adaptation 2-LAMA and the base-line BitTorrent [Objectives: 4,5; Chapters: 7,9]

- Jordi Campos, Nuria Piqué, Maite López-Sánchez, Marc Esteva "*Comparison of Topologies in Peer-to-Peer Data Sharing Networks*" Artificial Intelligence Research and Development, proceedings of the 13th International Conference of the Catalan Association for Artificial Intelligence. Edited by Rene Alquezar, Antonio Moreno, Josep Aguilar. Vol 220 of the series Frontiers in Artificial Intelligence and Applications 220. IOS Press. ISBN. 978-1-60750-642-3, ISSN 0922-6389. pp. 49-58, L'Espluga de Francolí (Tarragona, Spain), 20-22 Oct 2010.

  - It tested the heuristic approach of 2-LAMA when running on different network topologies, concluding that it was robust to these changes [Objective: 5; Chapters: 9]

- Jordi Campos, Maite Lopez-Sanchez, Marc Esteva "*A Case-Based Reasoning Approach for Norm Adaptation*" E. Corchado, M. Graña-Romay and A.M Savio (Eds). Springer. Hybrid Artificial Intelligence Systems part II. Lecture Notes in Artificial Intelligence, LNAI 6077, ISSN 0302-9743, ISBN 978-3-642-13802-7. 5th International Conference HAIS 2010, pp. 168-176, San Sebastián, Spain, June 2010.

  - It detailed a CBR approach to adapt norms in 2-LAMA, concluding it outperformed previous coordination models [Objectives: 4,5; Chapters: 7,9]

- Jordi Campos, Marc Esteva, Maite Lopez-Sanchez, Javier Morales, and Maria Salamó "*Organisational Adaptation of Multi-Agent Systems in a Peer-to-Peer scenario*" Journal Computing. ISSN: 0010-485X, Ed Springer (Wien). IF 2009 JCR: 1.033. Vol 91, pp 169-215 (DOI 10.1007/s00607-010-0141-9) Computing 2011.

  - This journal paper detailed several previous topics and enhanced the problem description, the 2-LAMA formalisation, the experimental methodology and the related work review [Objectives: 1,2,3,5; Chapters: 5,7,9]

- Jordi Campos, Maite Lopez-Sanchez, Marc Esteva, Maria Salamó and Pedro Ávila "*Learning Organisational Self-Adaptation in Multi-Agent Systems*" submitted to Journal of Systems and Software. Special issue Self-Adaptive Systems (under revision)

  - This journal paper is currently under revision. It includes the current CBR approach and some violation tests [Objectives: 4,5; Chapters: 7,9]

10.3 Future work

This thesis lays the foundations of our MAS architecture proposal (2-LAMA) to provide high-level coordination support services (assistance services in general, organisational adaptation in particular) to OCMAS that deal with regulation-oriented problems —e.g. the P2P sharing network. Nevertheless, along this work we identified several research branches that could be further explored.

First, this thesis focuses on a particular Assistance Layer service (adaptation), so the other proposed services –see §4.3– could be further elaborated. For instance, the Estimation service may alleviate agent development, since it is a way of abstracting some reasoning capabilities that could be provided by MAS infrastructure. Such an abstraction of high level functionalities may be a step further in software engineering.

Concerning the particular assistance service addressed in this thesis, the adaptation service, we suggested different alternatives –see §7.3.2– but there could be other ones. For instance, the norm adaptation could be performed using another machine learning technique like the reinforcement learning. Even more, using a hybrid learning approach could improve results. For example, we could merge mentioned reinforcement learning with our tailored CBR. Such a combination would require much more training than the CBR approach but could lead to current unexplored search space regions. Moreover, such a hybrid approach could decrease –or even terminate– CBR dependence on expert knowledge. Besides, it is also possible to go further in current CBR approach, like adding a maintenance phase to remove cases with a low evaluation value.

Further, this adaptation service could be provided by using another MAS architecture different from our proposal. That is to say, 2-LAMA was our proposal to guarantee the features of the agents in change of adaptation –see §5.4– but there may be other ways to guarantee such features. For example, one of the features of an adaptation agent is having access to gather certain information, which may depend on the other agents trusting it. Hence, some of the existing trust systems could be used to identify trusted third-parties among MAS participants, and let them act as the meta-level.

Moreover, although the 2-LAMA architecture is generic and abstract, we took some assumptions when specifying its components. For instance, we assumed that the adaptations are performed at a certain fixed frequency –see §5.5.4.2. In contrast, this frequency could vary depending on system outcomes stability. This way, if it is estimated that the system is not going to require any adaptation for a while, the next adaptation step could be delayed to avoid its computational and network costs. Even more, the meta-level coordination model could be modified by using a different agreement function based on consensus instead of majority —see §7.3.3.1. Also, the remote and local information could be aggregated in a different manner, instead of using the current weighted average —see §7.3.1.3. In fact, taking profit that 2-LAMA can be applied recursively –see §5.5.1– an additional meta-level could be in charge of these changes in current one.

Besides, current case study could be enhanced by approaching it more to real P2P sharing networks. For example, extending current protocol to deal with multiple-piece data —see §6.1. This may add more complexity, but could lead to get more out of 2-LAMA since the resources invested in the social structure phase would be exploited longer —see §9.4.1. Even focusing on current case study definition, it could have more norms, like one limiting the number of simultaneous sources. Currently, this limit is fixed to two simultaneous data sources –see §6.7.3– but it could vary depending on network status. Alternatively, other case studies could be chosen, like some of the other scenarios presented in §1.2.2.

Finally, regarding the empirical evaluation, certainly other experiments could be conducted to check any of the mentioned alternatives. But also, current exploration on open MAS issues showed that testing such concepts requires a very large number of simulations. Hence, some additional simulations could be done when having more time and computing resources. Even more, current parameter values could be modified to study their influence —with the corresponding increment of the number of simulations. Further, current entering/leaving and violation models could be enhanced, like using sanction or reward mechanisms —[Ávila et al., 2011] started this branch. In addition, there are other open MAS issues that could be essayed, like untrustworthy agent populations —e.g. in real P2P sharing networks, there are agents that send fake pieces of data in exchange of good pieces.

Overall, we think that the most promising future works are enhancing current learning method with reinforcement learning, continuing current open MAS exploration and developing the rest of assistance services.

Bibliography

[Aam, 1994] (1994). Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *Artificial Intelligence Communications*, 7:39–59.

[KIF, 1995] (1995). Knowledge Interchange Format (KIF). http://logic.stanford.edu/kif/specification.html.

[Bou, 2009] (2009). *Autonomic Electronic Institutions' Self-Adaptation in Heterogeneous Agent Societies*, volume 5368, pages 18–35. Springer.

[Arcos et al., 2005] Arcos, J. L., Esteva, M., Noriega, P., Rodríguez-Aguilar, J. A., and Sierra, C. (2005). Engineering open environments with electronic institutions. *Engineering Applications of Artificial Intelligence*, 18(2):191–204.

[Arcos et al., 2007] Arcos, J. L., Noriega, P., Rodríguez-Aguilar, J. A., and Sierra, C. (2007). E4mas through electronic institutions. In D. Weyns, H. P. and Michel, F., editors, *Environments for Multiagent Systems III*, volume 4389 of *Lecture Notes in Artificial Intelligence*, pages 184–202. Springer-Verlag.

[Argente et al., 2008] Argente, E., Botti, V., Carrascosa, C., Giret, A., Julian, V., and Rebollo, M. (2008). An Abstract Architecture for Virtual Organizations: The THOMAS project. Technical report, Grupo de Tecnología Informática - Inteligencia Artificial (GTI-IA), Universidad Politécnica de Valéncia. Technical Report.

[Artikis et al., 2009] Artikis, A., Kaponis, D., and Pitt, J. (2009). *Multi-Agent Systems: Semantics and Dynamics of Organisational Models*, chapter Dynamic Specifications of Norm-Governed Systems, pages 460–479. V. Dignum, IGI Global.

[Avouris and Gasser, 1992] Avouris, N. and Gasser, L. (1992). *Distributed artificial intelligence: Theory and praxis*, volume 5. Springer.

[Bellifemine et al., 2007] Bellifemine, F. L., Caire, G., and Greenwood, D. (2007). *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, NJ.

[Berns and Ghosh, 2009] Berns, A. and Ghosh, S. (2009). Dissecting self-* properties. In *2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 10–19. IEEE.

[BitTorrentInc., 2001] BitTorrentInc. (2001). BitTorrent protocol specification. http://www.bittorrent.org/beps/bep_0003.html.

[Boella et al., 2010] Boella, G., Pigozzi, G., Singh, M., and Verhagen, H. (2010). Normative multiagent systems: Guest editors' introduction. *Logic Journal of IGPL*.

[Boella and van der Torre, 2004] Boella, G. and van der Torre, L. (2004). Regulative and constitutive norms in normative multiagent systems. *Proceedings of KR'04*, pages 255–265.

[Bogdanovych et al., 2007] Bogdanovych, A., Esteva, M., Gu, N., Simoff, S., Maher, M., and Smith, G. (2007). The role of online travel agents in the experience economy. In *Proceedings of the 14th International Conference on Information Technology and Travel & Tourism (ENTER 2007). UK: Axon Imprint*, pages 81–91.

[Boissier and Gâteau, 2007] Boissier, O. and Gâteau, B. (2007). Normative multi-agent organizations: Modeling, support and control. In Boella, G., van der Torre, L., and Verhagen, H., editors, *Normative Multi-agent Systems*, number 07122 in Dagstuhl Seminar Proceedings, pages 1–17. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[Boissier et al., 2006] Boissier, O., Hübner, J. F., and Sichman, J. S. (2006). Organization oriented programming: From closed to open organizations. In O'Hare, G. M. P., Ricci, A., O'Grady, M. J., and Dikenelli, O., editors, *ESAW*, volume 4457 of *Lecture Notes in Computer Science*, pages 86–105. Springer.

[Bou et al., 2006] Bou, E., López-Sánchez, M., and Rodríguez-Aguilar, J. A. (2006). Norm adaptation of autonomic electronic institutions with multiple goals. *ITSSA*, 1(3):227–238.

[Bou et al., 2007] Bou, E., López-Sánchez, M., and Rodríguez-Aguilar, J. A. (2007). Self-adaptation in autonomic electronic institutions through case-based reasoning. In *Proceedings of MA4CS Workshop of ECCS'07*, page To appear as Lecture Notes in Computer Science.

[Busoniu et al., 2007] Busoniu, L., Babuska, R., and De Schutter, B. (2007). Multi-agent reinforcement learning: A survey. In *Control, Automation, Robotics and Vision, 2006. ICARCV'06. 9th International Conference on*, pages 1–6. IEEE.

[Cámara et al., 2006] Cámara, J. P., Gregori, M. E., Bada, G. A., García-Fornes, A., Julián, V., and Botti, V. J. (2006). Adding new communication

services to the FIPA message transport system. In *MATES*, volume 4196 of *LNCS*, pages 1–11.

[Campos et al., 2009a] Campos, J., López-Sánchez, M., and Esteva, M. (2009a). Assistance layer, a step forward in Multi-Agent Systems Coordination Support. In *Eighth International Conference on Autonomous Agents and Multiagent Systems*, pages 1301–1302.

[Campos et al., 2009b] Campos, J., López-Sánchez, M., and Esteva, M. (2009b). Assistance layer in a p2p scenario. In *Engineering Societies in the Agents World X (ESAW'09). Lecture Notes in Artificial Intelligence*, number 5881, pages 229–232. Spinger.

[Campos et al., 2010] Campos, J., López-Sánchez, M., Esteva, M., and Piqué, N. (to appear 2010). Comparison of Topologies in Peer-to-Peer Data Sharing Networks. In *Proceedings of the 13th International Conference of the Catalan Association for Artificial Intelligence (CCIA09).*

[Campos et al., 2008] Campos, J., López-Sánchez, M., Rodríguez-Aguilar, J. A., and Esteva, M. (2008). Norms in 2-LAMA. Technical Report RR-IIIA-2008-05, The Artificial Intelligence Research Institute (IIIA), Spanish National Research Council (CSIC).

[Campos et al., 2009c] Campos, J., López-Sánchez, M., Rodríguez-Aguilar, J. A., and Esteva, M. (2009c). Formalising situatedness and adaptation in electronic institutions. In *Coordination, Organizations, Institutions, and Norms in Agent Systems IV. Lecture Notes in Artificial Intelligence (LNAI)*, number 5428, pages 126–139. Springer.

[Cardoso and Oliveira, 2004] Cardoso, H. and Oliveira, E. (2004). Virtual Enterprise Normative Framework within Electronic Institutions. In *Proceedings of ESAW'04.*

[Cardoso et al., 2009] Cardoso, H., Oliveira, E., and Pesquisa, N. (2009). A Context-based Institutional Normative Environment. pages 140–155. Springer.

[Carley, 1995] Carley, K. (1995). Computational and mathematical organization theory: Perspective and directions. *Computational & Mathematical Organization Theory*, 1(1):39–56.

[Choffnes and Bustamante, 2008] Choffnes, D. and Bustamante, F. (2008). Taming the torrent: a practical approach to reducing cross-ISP traffic in peer-to-peer systems. *SIGCOMM Comput. Commun. Rev.*, 38(4):363–374.

[Chomsky, 1965] Chomsky, N. (1965). *Aspects of the Theory of Syntax*, volume 119. The MIT press.

[Common Logic Working Group, 2007] Common Logic Working Group (2007). Common Logic ISO/IEC IS 24707:2007.

[Corkill and Lesser, 1983] Corkill, D. D. and Lesser, V. R. (1983). The use of meta-level control for coordination in a distributed problem solving network. In *IJCAI'83: Proceedings of the Eighth international joint conference on Artificial intelligence*, pages 748–756, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Costa and Demazeau, 1996] Costa, A. and Demazeau, Y. (1996). Toward a formal model of multi-agent systems with dynamic organizations. In *Proceedings of the International Conference on Multi-Agent Systems, MIT Press, Kyoto, Japan*, page 431.

[de Almeida Júdice Gamito Dignum, 2004] de Almeida Júdice Gamito Dignum, M. V. F. (2004). *A model for organizational interaction: based on agents, founded in logic.* Universiteit Utrecht PhD Monography. Universiteit Utrecht, Faculteit Wiskunde en Informatica.

[de Pinninck et al., 2008] de Pinninck, A., Sierra, C., and Schorlemmer, M. (2008). Distributed Norm Enforcement Via Ostracism. *LNCS*, 4870:301.

[Demšar, 2006] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal Machine Learning Research*, 7:1–30.

[Dignum and Dignum, 2005] Dignum, V. and Dignum, F. (2005). Structures for agent organizations. In *International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, pages 215–220. IEEE.

[Dignum et al., 2005] Dignum, V., Dignum, F., Furtado, V., Melo, A., and Sonenberg, L. (2005). Towards a Simulation Tool for Evaluating Dynamic Reorganization of Agents Societies. In *Proc. of WS. on Socially Inspired Computing, AISB Convention*, volume 230, pages 153–162.

[Dijkstra, 1959] Dijkstra, E. W. (1959). A note on two problems in connection with graphs. *Numerische Mathematik*, 1:269–271.

[Duran et al., 2008] Duran, F., da Silva, V., and de Lucena, C. (2008). Using Testimonies to Enforce the Behavior of Agents. *LNCS*, 4870:218.

[Esteva, 2003] Esteva, M. (2003). *Electronic Institutions: from specification to development.* IIIA PhD. Vol. 19.

[Esteva et al., 2004] Esteva, M., Rosell, B., Rodriguez-Aguilar, J., and Arcos, J. (2004). Ameli: An agent-based middleware for electronic institutions.

[Ferber and Gutknecht, 1998] Ferber, J. and Gutknecht, O. (1998). A meta-model for the analysis and design of organizations in multi-agent systems. In Demazeau, Y., editor, *Proceedings of ICMAS'98*, pages 128–135. IEEE Computer Society.

[Ferber et al., 2004] Ferber, J., Gutknecht, O., and Michel, F. (2004). From Agents to Organizations: An Organizational View of Multi-agent Systems. In Giorgini, P., Müller, J. P., and Odell, J., editors, *Agent-Oriented Software Engineering IV*, pages 214–230. Springer.

[Ferber and Muller, 1995] Ferber, J. and Muller, J.-P. (1995). Influences and reaction: A model of situated multiagent systems. In Lesser, V., editor, *Proceedings of the First International Conference on Multi–Agent Systems*. MIT Press.

[Finin et al., 1997] Finin, T., Labrou, Y., and Mayfield, J. (1997). KQML as an agent communication language. In *Software Agents*, chapter 14, pages 291–316.

[FIPA, 1997] FIPA (1997). *FIPA 97 Specification Part 2: Agent Communication Language*. Foundation for Intelligent Physical Agents. http://www.fipa.org/spec/f8a22.zip.

[FIPA, 2001a] FIPA (2001a). *FIPA ACL Message Structure Specification*. Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa00061.

[FIPA, 2001b] FIPA (2001b). *FIPA Agent Discovery Service Specification*. Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa00095.

[FIPA, 2001c] FIPA (2001c). *FIPA Agent Management Specification*. Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa00023.

[FIPA, 2001d] FIPA (2001d). *FIPA Agent Message Transport Service Specification*. Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa00067.

[FIPA, 2001e] FIPA (2001e). *FIPA Communicative Act Library Specification*. Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa00037.

[FIPA, 2001f] FIPA (2001f). *FIPA Domains and Policies Specification*. Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa00089.

[FIPA, 2001g] FIPA (2001g). *FIPA English Auction Interaction Protocol Specification*. Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa00031.

[FIPA, 2001h] FIPA (2001h). *FIPA Ontology Service Specification*. Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa00086.

[FIPA, 2001i] FIPA (2001i). *FIPA SL Content Language Specification*. Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa00008.

[FIPA, 2002] FIPA (2002). *FIPA Abstract Architecture Specification*. Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa00001/.

[Foster et al., 2004] Foster, I., Jennings, N. R., and Kesselman, C. (2004). Brain meets brawn: Why grid and agents need each other. In *Proceedings of AAMAS'04*, pages 8–15.

[Frank Dignum, 2000] Frank Dignum, M. G. (2000). Issues in agent communication: An introduction. In *Issues in Agent Communication*, pages 1–16. Springer Berlin / Heidelberg.

[Friedman, 1937] Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701.

[Friedman, 1940] Friedman, M. (1940). A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, 11(1):86–92.

[Gaertner et al.,] Gaertner, D., García-Camino, A., Noriega, P., Rodríguez-Aguilar, J.-A., and Vasconcelos, W. Distributed norm management in regulated multi-agent systems. In *AAMAS'07*.

[Gaertner et al., 2009] Gaertner, D., Rodríguez-Aguilar, J., and Toni, F. (2009). Agreeing on institutional goals for multi-agent societies. *Coordination, Organizations, Institutions and Norms in Agent Systems IV*, pages 1–16.

[García-Camino et al., 2006] García-Camino, A., Rodríguez-Aguilar, J. A., Sierra, C., and Vasconcelos, W. (2006). A rule-based approach to norm-oriented programming of electronic institutions. *ACM SIGecom Exchanges*, 5(5):33–40.

[Gâteau et al., 2005] Gâteau, B., Boissier, O., Khadraoui, D., and Dubois, E. (2005). Moiseinst: An organizational model for specifying rights and duties of autonomous agents. In *EUMAS*, pages 484–485.

[Gou et al., 2007] Gou, X., Qin, N., and Li, Z. (2007). An Agent Communication Language for Multimedia Communication Task and Its Application. In *Mechatronics and Automation, 2007. ICMA 2007. International Conference on*, pages 1401–1405. IEEE.

[Grau et al., 2008] Grau, B., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., and Sattler, U. (2008). OWL 2: The next step for OWL. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(4):309–322.

[Gregori et al., 2006] Gregori, M. E., Cámara, J. P., and Bada, G. A. (2006). A jabber-based multi-agent system platform. In *Proceedings of AAMAS'06*, pages 1282–1284.

[Grizard et al., 2007] Grizard, A., Vercouter, L., Stratulat, T., and Muller, G. (2007). A peer-to-peer normative system to achieve social order. In *LNCS - Proc. of the Coordination, organizations, institutions, and norms in agent systems II*, volume 4386, page 274. Springer.

[Guessoum et al., 2004] Guessoum, Z., Ziane, M., and Faci, N. (2004). Monitoring and organizational-level adaptation of multi-agent systems. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 514–521, Washington, DC, USA. IEEE Computer Society.

[Gómez-Sanz et al., 2008] Gómez-Sanz, J. J., Fuentes-Fernández, R., Pavón, J., and García-Magariño, I. (2008). INGENIAS Development Kit: a visual multi-agent system development environment. In *Proceedings of AAMAS'08*, pages 1675–1676.

[Gómez-Sanz and Pavón, 2005] Gómez-Sanz, J. J. and Pavón, J. (2005). Implementing Multi-agent Systems Organizations with INGENIAS. In *Proceedings of ProMAS'05*, volume 3862, pages 236–251.

[Hewitt, 1986] Hewitt, C. (1986). Offices are open systems. *ACM Transactions on Office Information Systems*, 4(3):271–287.

[Hilpinen et al., 1971] Hilpinen, R., Kanger, S., Segerberg, K., and Hansson, B. (1971). *Deontic Logic: Introductory and Systematic Readings*. Reidel.

[Horling et al., 2001] Horling, B., Benyo, B., and Lesser, V. (2001). Using self-diagnosis to adapt organizational structures. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 529–536, New York, NY, USA. ACM.

[Horling and Lesser, 2004] Horling, B. and Lesser, V. (2004). A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(4):281–316.

[Hübner et al., 2002] Hübner, J., Sichman, J., and Boissier, O. (2002). A model for the structural, functional, and deontic specification of organizations in multiagent systems. *Advances in Artificial Intelligence*, pages 439–448.

[Hübner et al., 2004] Hübner, J. F., Sichman, J. S., and Boissier, O. (2004). Using the $\mathcal{M}$oise+ for a cooperative framework of mas reorganisation. In *LNAI - Proc. of the 17th Brazilian Symposium on Artificial Intelligence (SBIA'04)*, volume 3171, pages 506–515. Springer.

[Hübner et al., 2005] Hübner, J. F., Sichman, J. S., and Boissier, O. (2005). S-MOISE$^+$: A middleware for developing organised multi-agent systems. In *AAMAS Workshops*, volume 3913 of *LNCS*, pages 64–78. Springer.

[Iman and Davenport, 1980] Iman, R. and Davenport, J. (1980). Approximations of the critical region of the friedman statistic. *Communications in statistics*, pages 571–595.

[ISO 7498-1, 1994] ISO 7498-1 (1994). Information technology - Open Systems Interconnection - Basic Reference Model. http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip.

[Janert, 2009] Janert, P. K. (2009). *Gnuplot in Action: Understanding Data with Graphs*. Manning Publications Co., Greenwich, CT, USA.

[Jennings et al., 1998] Jennings, N., Sycara, K., and Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38.

[Jeon et al., 2002] Jeon, H., Petrie, C., and Cutkosky, M. (2002). JATLite: a Java agent infrastructure with message routing. *Internet Computing, IEEE*, 4(2):87–96.

[Jones and Goel, 2004] Jones, J. and Goel, A. K. (2004). Revisiting the Credit Assignment Problem. In *Challenges of Game AI: Proceedings of the AAAI*, volume 4, page 4.

[Kephart and Chess, 2003] Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *IEEE Computer*, 36(1):41–50.

[Kitio et al., 2008] Kitio, R., Boissier, O., Hübner, J. F., and Ricci, A. (2008). Organisational artifacts and agents for open MAS organisations. In *Proceedings of COIN at AAMAS'08*, volume 4870, pages 171–186.

[Kota et al., 2009] Kota, R., Gibbins, N., and Jennings, N. (2009). Decentralised structural adaptation in agent organisations. In *AAMAS Workshop on Organised Adaptation in Multi-Agent Systems, Estoril, Portugal*, pages 54–71. Springer.

[Lemaître and Excelente, 1998] Lemaître, C. and Excelente, C. (1998). Multi-agent organization approach. In *Proceedings of II Iberoamerican Workshop on DAI and MAS*.

[Lesser et al., 2004] Lesser, V., Decker, K., Wagner, T., Carver, N., Garvey, A., Horling, B., Neiman, D., Podorozhny, R., Prasad, M., Raja, A., et al. (2004). Evolution of the GPGP/TAEMS domain-independent coordination framework. *Autonomous Agents and Multi-Agent Systems*, 9(1):87–143.

[Lewis, 1969] Lewis, D. (1969). *Convention: A Philosophical Study*. Harvard University Press.

[Molesini et al., 2007] Molesini, A., Denti, E., and Omicini, A. (2007). From aose methodologies to mas infrastructures: The soda case study. In *Engineering Societies in the Agents World 2007*, pages 283–298.

[Mukherjee et al., 2008] Mukherjee, P., Sen, S., and Airiau, S. (2008). Norm emergence under constrained interactions in diverse societies. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 779–786, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Nemenyi, 1963] Nemenyi, P. B. (1963). *Distribution-free Multiple Comparisons*. PhD thesis, Princeton University.

[North et al., 2005] North, M., Howe, T., Collier, N., and Vos, J. (2005). Repast Simphony Runtime System. In *Agent Conference on Generative Social Processes, Models, and Mechanisms*.

[Novo and Campos, 2010] Novo, A. and Campos, J. (2010). Simulación de redes Peer-to-Peer con OpenMas. Master's thesis, Universitat de Barcelona.

[Okouya and Dignum, 2008] Okouya, D. M. and Dignum, V. (2008). OperettA: A prototype tool for the design, analysis and development of multi-agent organizations. In *Proceedings of the 7th international joint conference on Autonomous Agents and MultiAgent Systems (AAMAS'08): demo papers*, pages 1677–1678. International Foundation for Autonomous Agents and Multiagent Systems.

[Omicini, 2001] Omicini, A. (2001). SODA: societies and infrastructures in the analysis and design of agent-based systems. In *Proceedings of AOSE'00*, pages 185–193.

[Omicini et al., 2004] Omicini, A., Ossowski, S., and Ricci, A. (2004). Coordination infrastructures in the engineering of multiagent systems. *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, 11:273–296.

[Omicini et al., 2008] 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.

[Padgham and Winikoff, 2005] Padgham, L. and Winikoff, M. (2005). *Agent-Oriented Methodologies*, chapter 5. Prometheus: A Practical Agent-Oriented Methodology. Idea Group.

[Panait and Luke, 2005] Panait, L. and Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434.

[Pavón and Gómez-Sanz, 2003] Pavón, J. and Gómez-Sanz, J. J. (2003). Agent oriented software engineering with INGENIAS. In *Procedings of CEECMAS*, volume 2691, pages 394–403.

[Plaza and McGinty, 2006] Plaza, E. and McGinty, L. (2006). Distributed case-based reasoning. *The Knowledge engineering review*, 20(03):261–265.

[Pujol et al., 2005] Pujol, J., Delgado, J., Sanguesa, R., and Flache, A. (2005). The role of clustering on the emergence of efficient social conventions. In *IJCAI'05: Proceedings of the 19th international joint conference on Artificial intelligence*, pages 965–970.

[Ricci et al., 2006] Ricci, A., Viroli, M., and Omicini, A. (2006). CArtAgO:A framework for prototyping artifact-based environments in MAS. In *Proceedings of E4MAS'06*, volume 4389 of *LNCS*.

[Riesbeck and Schank, 1989] Riesbeck, C. K. and Schank, R. C. (1989). *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, Hillsdale, NJ, US.

[Russell et al., 1995] Russell, S., Norvig, P., Canny, J., Malik, J., and Edwards, D. (1995). *Artificial intelligence: a modern approach*. Prentice hall Englewood Cliffs, NJ.

[Sabater-Mir, 2006] Sabater-Mir, J. (2006). Towards the next generation of computational trust and reputation models. In *Modeling Decisions for Artificial Intelligence*, volume 3885, pages 19–21. Springer.

[Salamó and López-Sánchez, 2011] Salamó, M. and López-Sánchez, M. (2011). Rough set based approaches to feature selection for case-based reasoning classifiers. *Pattern Recogn. Lett.*, 32:280–292.

[Salazar-Ramirez et al., 2008] Salazar-Ramirez, N., Rodríguez-Aguilar, J. A., and Arcos, J. L. (2008). An infection-based mechanism for self-adaptation in multi-agent complex networks. In Brueckner, S., Robertson, P., and Bellur, U., editors, *2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2008*, pages 161–170.

[Saunier et al., 2006] Saunier, J., Balbo, F., and Badeig, F. (2006). Environment as active support of interaction. In Weyns, D., Parunak, H. V. D., and Michel, F., editors, *E4MAS*, volume 4389 of *Lecture Notes in Computer Science*, pages 87–105. Springer.

[Savarimuthu and Cranefield, 2009] Savarimuthu, B. T. R. and Cranefield, S. (2009). A categorization of simulation works on norms. (09121).

[Savarimuthu et al., 2008] Savarimuthu, B. T. R., Cranefield, S., Purvis, M., and Purvis, M. (2008). Role model based mechanism for norm emergence in artificial agent societies. In *Proceedings of the 2007 international conference on Coordination, organizations, institutions, and norms in agent systems III*, COIN'07, pages 203–217, Berlin, Heidelberg. Springer-Verlag.

[Searle, 1995] Searle, J. (1995). *The Construction of Social Reality*. The Free Press, New York.

[Searle, 1969] Searle, J. R. (1969). *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press.

[Sen and Sen, 2010] Sen, O. and Sen, S. (2010). Effects of Social Network Topology and Options on Norm Emergence. *Coordination, Organizations, Institutions and Norms in Agent Systems V*, pages 211–222.

[Sen and Airiau, 2007] Sen, S. and Airiau, S. (2007). Emergence of norms through social learning. In Veloso, M. M., editor, *IJCAI*, pages 1507–1512.

[Serugendo et al., 2006] Serugendo, G. D. M., Gleizes, M.-P., and Karageorgos, A. (2006). Self-Organisation and Emergence in MAS: An Overview. *Informatica*, 30:45–54.

[Sharma, 2010] Sharma, A. (2010). *Introduction To Set Theory*. DPH mathematics series. Discovery Publishing House.

[Sichman et al., 2006] Sichman, J. S., Boissier, O., and Hübner, J. F. (2006). Organization oriented programming: From closed to open mas. Presented at Iberagents.

[Sierra et al., 2007] Sierra, C., Rodríguez-Aguilar, J. A., Noriega, P., Esteva, M., and Arcos, J. L. (2007). personal communications about Electronic Institutions.

[Sierra et al., 2006] Sierra, C., Thangarajah, J., Padgham, L., and Winikoff, M. (2006). Designing institutional multi-agent systems. In *AOSE'06*, volume 4405 of *LNCS*, pages 84–103.

[Sims et al., 2008] Sims, M., Corkill, D., and Lesser, V. (2008). Automated Organization Design for Multi-agent Systems. *Autonomous Agents and Multi-Agent Systems*, 16(2):151–185.

[Smith, 1982] Smith, B. C. (1982). *Reflection and Semantics in a Procedural Language*, volume MIT/LCS/TR-272. MIT Laboratory for Computer Science Technical Report.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*. The MIT press.

[Sycara et al., 2003] Sycara, K. P., Paolucci, M., Velsen, M. V., and Giampapa, J. A. (2003). The RETSINA MAS infrastructure. *Autonomous Agents and Multi-Agent Systems*, 7(1-2):29–48.

[Valckenaers et al., 2007] Valckenaers, P., Sauter, J. A., Sierra, C., and Rodríguez-Aguilar, J. A. (2007). Applications and environments for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):61–85.

[Vallee et al., 2005] Vallee, M., Ramparany, F., and Vercouter, L. (2005). A Multi-Agent System for Dynamic Service Composition in Ambient Intelligence Environments. In *Advances in pervasive computing*, pages 8–11.

[Verma, 2002] Verma, D. (2002). *Content distribution networks*. Wiley Online Library.

[Weyns and Holvoet, 2004] Weyns, D. and Holvoet, T. (2004). A formal model for situated multi-agent systems. *Fundam. Inform*, 63(2-3):125–158.

[Wooldridge, 2009] Wooldridge, M. (2009). *An introduction to multiagent systems*. Wiley.

[Wooldridgey and Ciancarini, 2001] Wooldridgey, M. and Ciancarini, P. (2001). Agent-oriented software engineering: The state of the art. In *Agent-Oriented Software Engineering*, pages 55–82. Springer.

[Xie et al., 2008] Xie, H., Yang, Y. R., Krishnamurthy, A., Liu, Y., and Silberschatz, A. (2008). P4P: provider portal for applications. *ACM SIGCOMM Computer Communication Review*, 38(4):351–362.

[Zambonelli et al., 2003] Zambonelli, F., Jennings, N. R., and Wooldridge, M. (2003). Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317–370.

[Zhang et al., 2008] Zhang, C., Abdallah, S., and Lesser, V. (2008). MASPA: Multi-Agent Automated Supervisory Policy Adaptation. Technical Report 03.

[Zhang et al., 2009] Zhang, C., Abdallah, S., and Lesser, V. (2009). Integrating Organizational Control into Multi-Agent Learning. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 757–764. International Foundation for Autonomous Agents and Multiagent Systems.

[Ávila et al., 2011] Ávila, P., López-Sánchez, M., and Esteva, M. (2011). On norm compliance within organization centered multi-agent systems applied to a Peer-to peer sharing network scenario. Master's thesis, Universitat Politècnica de Catalunya.