Agent-Oriented Software Engineering for Internet Applications

Franco Zambonelli[1] Nicholas R. Jennings[2] Andrea Omicini[3] Michael Wooldridge[4]

[1] Dipartimento di Scienze dell'Ingegneria
Università di Modena e Reggio Emilia
Via Campi 213-b – 41100 Modena, Italy
franco.zambonelli@unimo.it

[2] Department of Electronics and Computer Science
University of Southampton, Southampton SO17 1BJ, United Kingdom
nrj@ecs.soton.ac.uk

[1] Dipartimento di Elettronica, Informatica e Sistemistica
Università di Bologna
Viale Risorgimento 2 – 40126 Bologna, Italy
aomicini@deis.unibo.it

[4] Department of Computer Science, University of Liverpool
Liverpool L69 7ZF, United Kingdom
M.J.Wooldridge@csc.liv.ac.uk

July 4, 2000

Abstract

The metaphors of *autonomous agents* and *agent societies* have the potential to make a significant impact on the processes of analysis, design, and development of complex software systems on the Internet. In this chapter, we concentrate predominantly on agent societies, and show how work on coordination models and technologies provides a powerful framework for the engineering of Internet-based, multi-agent systems. First, we introduce the concepts of agent, multi-agent system, and agent-oriented software engineering, and highlight the specific issues that arise when we take the Internet as the environment that agents inhabit. We then provide a brief survey of the state of the art in the area of agent-oriented methodologies, paying particular attention to the Gaia methodology for agent-oriented analysis and design. Gaia was originally conceived for benevolent agents inhabiting closed systems. However, to broaden its scope, we show how insights from the area of coordination models can be incorporated in order to make it more suitable for developing Internet-based applications.

Published as Chapter 13 in the Book: *Coordination of Internet Agents: Models, Technologies and Applications*, A. Omicini, F. Zambonelli, M. Klusch, R. Tolksdorf (Eds.), Springer, 2000.

1 Introduction

Agent-based computing [31, 19, 34] is rapidly emerging as a powerful technology for the development of complex software systems, synthesising contributions from many different research areas, including artificial intelligence, software engineering, robotics, and distributed computing. Developing applications in terms of *autonomous software agents* that exhibit proactive and intelligent behaviour, and that interact with one another in terms of high-level protocols and languages, leads to a new programming paradigm. This paradigm, which we term *agent-oriented software engineering*, is well suited to tackling the complexity of today's software systems. Firstly, today's software systems are required to deal with complex and unpredictable execution environments which object-oriented languages and applications seem unable to model and tackle in a natural and clean way [2]. The Internet is probably the most complex and unpredictable environment that application designers have to face today, and Internet applications are arguably the most important area in which the agent paradigm can fulfill its potential. Secondly, the abstraction level defined by the agent-based approach is higher than that of the object-oriented paradigm, as the behaviour of agents more closely reflects that of the human beings whose work they are delegated to perform and/or support.

By dint of being a new programming paradigm, the development of agent-based applications implies new programming abstractions and techniques. This necessity is born out by the significant body of research that has been undertaken in defining and developing new models and systems for building agent-based, and multi-agent, systems. These efforts include problems related to the definition of suitable architectures for agents [23], agent communication languages [11, 10], coordination model for agents [28, 6], as well as models for the specification and verification of multi-agent systems [3]. However, the development of complex multi-agent systems requires not only new models and technologies, but also new *methodologies* to support developers in an engineered approach to the *analysis* and *design* of such systems. This need is especially pressing if multi-agent systems are to break out of being a niche technology used by the few, to something that is part of the armoury of a mainstream software engineer.

In the last few years, several attempts have been made to develop *agent-oriented modelling techniques and methodologies* [18, 20]. However, none of these techniques can yet be regarded as a comprehensive methodology for the analysis and design of multi-agent systems. The *Gaia* methodology [35] is one of the few attempts that is specifically tailored to the analysis and design of multi-agent systems and that deals with both the micro (intra-agent) level and the macro (inter-agent) level of analysis and design. However, Gaia, as it presently stands, is not a general methodology for all kinds of multi-agent system. Rather, it is intended to support the development of distributed problems solvers in which the system's constituent components are known at design time (i.e., a closed system) and in which all agents are expected to cooperate towards the achievement of a global goal. For these reasons, Gaia is not suitable for the analysis and design of Internet applications, where openness and self-interest are key factors.

One of the main reasons why most extant agent-oriented methodologies (including Gaia) are ill suited to open systems are that they conceive all interactions as occurring directly between agents. This makes it difficult to enforce any form of control on interactions as required, for example, to handle the arrival of new agents into the system or to constrain the actions of possibly self-interested agents. From a more general perspective, these methodologies fail to adequately address the concept of "agent societies". Agents need to be conceived as populating a society, whose global activities are expected to proceed according to specific social laws and conventions. In other words, the proper functioning of a multi-agent system usually relies on some form of global laws that the agents living in it have to obey when interacting with other agents. For example, the correct functioning of agent-mediated auctions require participating agents to comply to the conventions that rule each specific auction type [26].

In this chapter, we propose the adoption of *coordination models* [15, 8] as the conceptual abstraction that can be exploited in the analysis and design of multi-agent systems for the Internet. Furthermore, we sketch how a coordination model can be exploited to make a methodology, such as Gaia, suitable for Internet applications. The key idea stems from the fact that a coordination model abstracts the concept of coordination media, intended as the place where interactions occur, and of coordination laws, intended as the rules that the coordination media enforces in interactions. Therefore, in general, a coordination model could be exploited so as to act as the repository of the social laws of an application, which all agents, having to interact with each other through the mediation of the coordination media, are forced to respect. More specifically, a coordination model could be exploited as a mediator (so as to enable interactions between entities that do not know one another), as well as a global controller for all interactions (so as to control and—if necessary—constrain, the behaviour of possibly self-interested agents).

The remainder of this chapter is organised as follows. Section 2 discusses the basic characteristics of the agent paradigm and of multi-agent systems, and motivates the need for software engineering methodologies for multi-agent systems. Section 3 analyses a number of methodologies for agent-oriented software engineering (paying particular attention to Gaia) and shows the limitation of these methodologies when dealing with Internet applications. Section 4 shows how a coordination model can be used as a powerful framework for the analysis and design of multi-agent systems on the Internet. On this base, section 5 sketches a new methodology for the analysis and design of multi-agent systems on the Internet. Section 6 concludes the chapter and outlines open issues and research directions in the area.

2 Engineering Multi-Agent Systems on the Internet

2.1 What is a multi-agent system?

The very notion of *agent* is one of the most debated and controversial in the fields of Computer Science and Artificial Intelligence. According to [34], an agent is characterised by *autonomy*, *social ability*, *reactivity*, and *pro-activeness*. Since we are mainly interested in Internet agents, here, we may think of an agent as an autonomous software entity which interacts with its environment (the Internet) and with other agents pro-actively (that is, by its own initiative) in order to achieve its own goals (typically accomplishing some information gathering or dissemination task).

These agents typically represent different users and thus there are many of them in

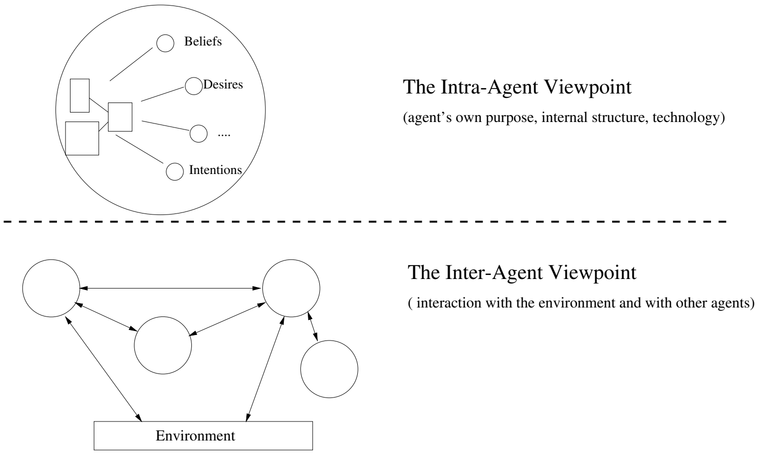

Figure 1: Intra-agent versus inter-agent viewpoints

a given environment. Thus, *multi-agent systems* can be considered as ensembles of autonomous agents, acting and working independently from each other, each representing an independent *locus* of control of the whole system. Each agent tries to accomplish its own task(s) and, in so doing, will typically need to interact with other agents and its surrounding environment in order to obtain access to information/services that it does not possess or to coordinate its activities to ensure its goals can be met. In many cases, however, a multi-agent system, as a whole, achieves more complex or wider goals than the mere sum of its component agent's goals. As an example, in a system composed of autonomous agents, each looking for computational resources to exploit, a global load balancing of the activities can be achieved, despite the fact that no single agent explicitly worries about load balancing. As another example, in systems for agent-mediated auctions, the opposing forces of buyers and sellers (the former attempting to sell a good at a high-price, the latter attempting to buy it at a low-price) can naturally lead to a reasonable pricing of goods.

The above perspective typically leads to the conception of a multi-agent system as a *society* of agents, where the mutual interactions between agents and with their environment leads to a useful global behaviour. Correspondingly, each agent of a multi-agent system may be observed according to two different perspectives (see Figure 1): from the inside, as an individual (software) system, with its own purpose/structure/technology (the *intra-agent* viewpoint), and from the outside, as part of a society, interacting with other individuals, accessing resources, and exploiting the social infrastructure (the *inter-agent* viewpoint). The latter aspect is what makes a multi-agent system something of radical departure from more traditional (software) systems, and is therefore the main topic of interest in this chapter.

More precisely, we may describe a multi-agent system by abstracting three fundamental concepts: *(i)* the agents, taken as individuals, *(ii)* the interactions among agents, like communication acts [10, 11], and *(iii)* the mutual dependencies between agents and social relationships, or *organisational relationships* [35, 9]. Clearly, the aspects related to the inter-agent viewpoint are represented by the last two concepts and characterise a multi-agent system as a society of interacting individuals. In particular, we may think of *(ii)* and *(iii)* as two different levels of social interaction. Speech acts and agent communication languages

[32, 10, 11] constitute the basic elements of interaction, while organisational relationships provide the conceptual framework upon which each agent finds a well-specified position (or *role*) in the society, which motivates and structures the interactions among the agents.

It is the organisation of agents that gives a multi-agent system more than the knowledge, competence, and abilities provided by all its individual agents, and represents the value-added of the multi-agent approach. In fact, it is possible to change the behaviour of a multi-agent system without changing the constituent agents merely by altering the organisational relationships between them. Generally speaking, this means that the very essence of a multi-agent system implies that the organisational structures, that is, the societies of agents, have to be taken into account as *first-class entities*, and analysed, designed and implemented as such.

2.2 Engineering a multi-agent system

In seeking to develop an agent-oriented methodology, the most obvious point of departure is to consider object-oriented methodologies [2]. These methodologies have gained widespread acceptance and use. In particular, the emphasis of such techniques on both encapsulation and interaction among entities suggests that a multi-agent system could actually be built by exploiting an object-oriented methodology [27, 5, 30].

However, object-oriented approaches fall short in supporting the full notions of agents , and organisations as first-class entities [12, 20]. This is hardly surprising since these represent the two main distinguishing features of multi-agent systems with respect to object-based systems. Object-oriented methodologies have objects as their basic conceptual model and therefore have no support for capturing reactive and proactive behaviour, nor for maintaining a balance between the two. Additionally, there is no first class notion of an *organisational structure* in the object-oriented world. Organisation structure in object-oriented systems is implicit within `public` and `private` qualifiers, and the "handles" that agents have on one-another. Where entities in systems represent real-world entities that do have organisational relationships to one-another, this level of representation is inadequate.

Consequently, we believe what is needed is the definition of suitable engineering methodologies that are specific for multi-agent systems. These need to go beyond traditional object-oriented approaches by providing explicit support for the notions of agenthood and agent society.

2.3 Agent-oriented analysis and design

As with traditional methodologies, agent-oriented analysis starts from the definition of a system's requirements and goals. The application domain is studied and modelled, the computational resources available and the technological constraints are listed, the fundamental application goals and targets are devised, and so on. Then, the application's global goals are typically decomposed into smaller sub-goals, until they become manageable at the abstraction level provided by the approach adopted—which is when the design phase comes in.

In particular, when dealing with the analysis of an agent, we are no longer limited to defining computable functionalities, as in procedure-oriented methodologies, or sets of related services, to be encapsulated into an object or a component. Instead, since, by definition, agents have goals which they pursue pro-actively and in an autonomous way, agent-oriented analysis must identify the *responsibilities* of an agent. That is, the activities it has to carry out to perform one or more *tasks*, which may include specific *permissions* to access and influence the surrounding environment, as well as *interactions* with the other agents in the application. Design is concerned with the representation of the abstract models resulting from the analysis phase in terms of the design abstractions provided by the methodology. In agent-oriented software engineering, this means responsibilities, tasks, and interaction protocols, should be mapped onto agents, high-level interactions and organisations.

When assigned to an agent, a task defines both its inner structure, in terms of its environment representation and capabilities (from the competence required by the task), and its interaction protocol (roughly, from the information required and provided and the actions to be performed to accomplish the task). More precisely, an agent will be designed so as to be able to represent that portion of the environment with which it is concerned (its *sphere of influence*), to infer new information from this knowledge, and to act somehow on the environment. Moreover, its interaction protocols will be defined so as to enable the agent to contact other agents, get from them the information it needs, transmit newly inferred information to them, and coordinate its activities with the other agents.

One may think that "grown-up" objects may be charged with responsibilities, tasks and complex interaction protocols towards a useful representation of agents. However, this perspective misses what is peculiar to multi-agent systems engineering, that is, the concept of an *agent society*, structured according to organisational relationships. When agents have to live in an organised society, the analysis phase should identify not only the responsibilities of the agents as individuals, (i.e., their *individual tasks*), but also their global responsibilities to the multi-agent system as a whole, (i.e., its *social tasks*).

Individual tasks are associated with one specific competence in the system, related to the need to access and effect a specific portion of the environment and carry out some simple task. Each agent in the system is assigned one or more individual tasks, and the agent assumes full responsibility for carrying out assigned tasks. From an organisational perspective, this corresponds to assigning each agent a specific role in the organisation/society. Of course, multiple agents in a system can be assigned the same task/role.

In contrast, social tasks represent the global responsibilities of the agent system. In order to carry out such tasks, several, possibly heterogeneous competences, will usually need to be combined. The achievement of social tasks leads to the identification of global *social laws* that have to be respected and/or enforced by the society of agents, to enable the society itself to function properly and accordingly to the global expected behaviour.

To clarify the above concepts, let us consider the problem of engineering a multi-agent software system to manage the review process for an international conference. An individual task that easily comes out from the analysis phase is picking up a given number of papers, and taking the responsibility for reviewing them. The task can be defined as an individual one in that it can be easily and naturally charged to a single agent, representing a referee. A social task, instead, could be to ensure that each submitted paper has at least three reviews, by three different referees of different institutions. This task requires that a given number of entities have the competence for reviewing papers, and that they can assume the responsibility for such a task (an individual one). However, the task also requires that the reviewing competence and tasks are coordinated and organised so as to ensure the required number and variety of reviews. Therefore, the task cannot naturally be committed to a single agent, but its achievement involves several of the agents in the system.

According to the above considerations, any methodology for agent-oriented software engineering must provide suitable abstractions and tools to model not only the individual tasks of agents, but also the social ones.

2.4 Engineering systems for the Internet

The choice of the Internet as the target environment for the multi-agent system has a number of implications for the engineering of the system.

The first consequence of such a choice arises from the nature of the Internet as a global, distributed, and heterogeneous *information system*. Many of the activities of the agents are concerned with accessing, understanding, relating, modifying, and producing information. Apart from the abilities required of the individual agents, this makes it easy and natural to design multi-agent systems as information-based systems, where *all* the activities are interpreted and modelled in terms of the information required or made available. In particular, this also applies to the interactions of the agents. As a result, methodologies supporting this view of multi-agent systems naturally fit their engineering on the Internet.

Another feature which is intrinsic to the Internet is *openness*. One may easily envision a (not so far) future where the Internet will host thousands of multi-agent systems, some of which interact directly or indirectly, with agents moving from one multi-agent system to another. This makes it difficult to use methodologies that assume the multi-agent system is *closed*—that is, that the number and the nature of the agents are known once and and for all a priori. Rather, the ability to support *openness*—as a feature, rather than as a problem—seems to be a mandatory requirement for any methodology for Internet-based multi-agent systems. This obviously holds for other typical Internet features, too: unpredictability, dynamicity, and unreliability. In particular, this is relevant for large-scale multi-agent systems, where the absence of centralised control makes it impossible to have a complete and reliable static picture of the agent's environment, on which analysis and design might be grounded on.

Furthermore, the properties of openness and unpredictability mean it is simply not feasible to make the cooperativeness of agents a basic assumption in the analysis and design phases. Some agents may indeed be cooperative, accepting goals on behalf of other agents, for instance, or agreeing on some protocol for cooperative interaction. However, many other agents will be purely *self-interested* and will pursue their own goals.

A methodology for agent-based Internet applications should therefore be able to support the engineering of MAS that incorporates not only well-known agents with a cooperative attitude, but also previously unknown agents that dynamically join the system (i.e., the agent society) and that are likely to exhibit self-interested behaviour.

3 Software Engineering Methodologies for MAS

This section provides a brief overview of the state of the art in the area of software engineering methodologies for MAS. In particular, we detail the main characteristics of the

Gaia methodology and outline its limitations with respect to dealing with Internet-based applications.

3.1 State of the art

A number of different methodologies have been proposed in recent years for modelling and engineering agents and multi-agent systems (see [18] for a survey).

As already observed, traditional methodologies for analysis and design are poorly suited for multi-agent systems because of the fundamental mismatch between the respective levels of abstraction. Despite this mismatch, however, several proposals do take object-oriented modelling techniques or methodologies as their basis. On the one hand, some proposals directly extend the applicability of object-oriented methodologies and techniques to the design of agent systems. For example, some proposals attempts to directly apply the UML notation to represent agent systems and their patterns of interaction [22, 33]. However, these proposals fail to capture the autonomous and proactive behaviour of agents, as well as the richness of their interactions. On the other hand, some proposals seek to extend and adapt object-oriented models and techniques to define a methodology for use in multi-agent systems. This can lead, for example, to extended models for representing agent behaviour and their interactions [5, 23, 30], as well as to agent-tuned extensions of UML [27, 30]. However, although these proposals can sometimes achieve a good modelling of the autonomous behaviour of agents and of their interactions, they lack the conceptual mechanisms for adequately dealing with organisations and agent societies.

A different set of proposals build upon and extend methodologies and modelling techniques from knowledge engineering [3, 4, 16]. These techniques provide formal and compositional modelling languages for the verification of system structure and function. These approaches are well-suited to modelling knowledge- and information- oriented agents (as are found in several Internet applications). However, since these approaches usually assume a centralised view of knowledge-based systems, they fail to provide adequate models and support for the societal view of multi-agent systems. The CommonKads approach [17] attempts to overcome these limitations by explicitly introducing into the methodology the abstraction of agent society. However, this simply reduces to modelling a society as a collection of interacting entities, with no identification of concepts such as social tasks or social laws.

Other models and approaches attempt to model and implement multi-agent systems from an "organisation-oriented" point of view [12]. These help pave the way towards agent-oriented methodologies by explicitly conceiving multi-agent systems as organisations [9] or as societies. However, these proposals define an organisation merely as a collection of interacting roles, thus failing, again, to deal with the key point of social tasks.

3.2 The Gaia methodology

Gaia is a methodology for agent-oriented analysis and design that makes explicit use of an organisational point of view. In Gaia, analysis and design are well-separated phases (see Figure 2). Analysis aims to develop an understanding of the system and its structure, in terms of the roles that have to be played in the agent organisation and of their interactions, without any reference to implementation details. The design phase aims to define the actual

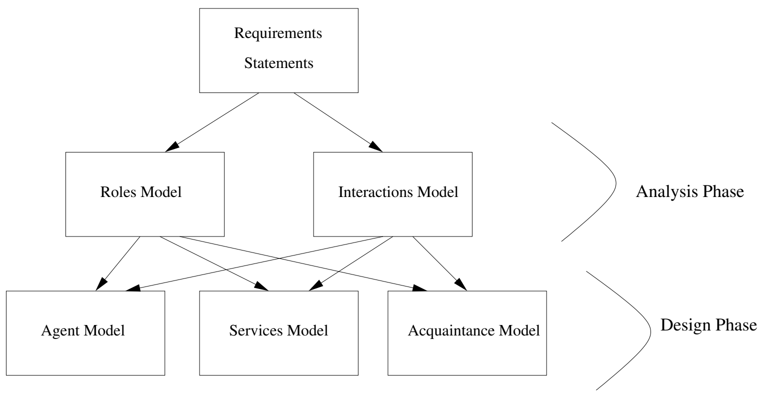

Figure 2: The basic concepts of the Gaia methodology

structure of the agent system, in terms of the agent classes and instances composing the system, of the services to be provided by each agent, and of the acquaintances' structure.

In more detail, the analysis phase aims to identify what the actual organisation of the multiple agents should look like. It does this by decomposing the system into abstract "loci of control"; i.e., the *roles* to be played in the organisation, and the way in which they interact accordingly to specific *protocols* (this, respectively, defines the *roles model* and *interactions model*). In particular, Gaia suggests the following steps:

1. Identify the roles in the system (these typically correspond to individuals within an organisation, departments, or organisations themselves) and define a list of the key roles in an informal description language.

2. For each role, identify the associated protocols, i.e., the patterns of interaction that are likely to occur between roles.

3. Elaborate the complete roles model and interactions model and, if required, iterate the previous stages.

The expected outputs of the analysis phase are a fully elaborated roles model—describing each role in terms of responsibilities, permissions, interaction protocols, and activities—and an interactions model—describing each protocol in terms of data exchanged and partners involved.

The design phase starts from the models defined during the analysis phase and aims to define the actual agent system in such a way that it can easily be implemented. To this end, the design phase has to decide which classes of agents (and how many) have to play the roles identified during the analysis phase, which services agents must provide to fulfill their role, and what is the actual topology of the interactions that flows from the interaction and the agent models. In more detail, the design phase involves the following stages:

1. Identify the agent model, that is, aggregate roles into agent types and refine, to form an agent type hierarchy, and evaluate the number of required instances for each class;

2. Identify the services that agents have to provide in order to fulfill their assigned roles, by analysing the activities and responsibilities, as well as the protocols defined by each of the roles;

3. Develop the acquaintance model, to identify inefficiencies in the design and, if required, iterate the previous stages.

The expected output of the design phase is the actual architecture (i.e., organisation) of the agent system, which can then be implemented using more traditional techniques (such as object-orientation and component-ware).

3.3 Applying Gaia

To illustrate the principles of the Gaia methodology, we will consider how it can be applied to the task of developing a software system to support the task of managing the review process of an international conference [7]. Once the submission deadline has passed, the program committee (PC) has to handle the review of the papers; contacting potential referees and asking them to review some of the papers by a given deadline. After a while, reviews come in and they are then used to decide about the acceptance/rejection of the submissions. The process is essentially a workflow: the activities of a number of different individuals have to be synchronised, data has to be exchanged, and each person involved may be required to perform different activities depending on the global status of the process. The whole process can be effectively supported by means of a multi-agent system (as in ADEPT [21]), which assists with workflow management.

Depending on the size of the conference (i.e., the number of submissions it receives), different ways of handling the review process can be adopted. Let us consider the case of a large conference, in which the PC Chair does not handle the assignment of individual papers to referees and in which the PC Members need to recruit external referees in order to assist them with their reviewing task. In this organisation, the PC Chair can simply partition the submitted papers and then send one of the partitions to each of the PC members. The duty of each PC Member is then to find three different referees for each of the papers in their assigned partition, collect the corresponding reviews, and send them back to the PC Chair. The PC Chair eventually collects and ranks the reviews.

In Gaia, the first step of the analysis phase is to identify the roles involved. In this case, the following roles are easy to identify:

- *paper partitioner*: in charge of partitioning the papers and assigning the partitions to PC members according to some criterion (for example, based on load balancing or competences of PC Members);

- *review allocator*: in charge of finding three reviews for each of the papers in a given partition, and collating the reviews that are returned;

- *reviewer*: in charge of receiving a paper and reviewing it by the stated deadline;

- *review collector*: in charge of collecting all the reviews, ranking them and deciding upon acceptance and rejection.

For each of these roles, the associated permissions, activities, and responsibilities need to be identified. The role reviewer, for example, requires the permission to read the paper it is assigned for review and the permission to write the corresponding review form. Its activities involve receiving a paper to review, reviewing it in due time, and sending the completed review form back. Its main responsibility is to ensure that the completed review form is sent back on time.

The interaction model follows very naturally from the definition of the roles. It basically amounts to specifying which protocols involve which role pairings and what information is exchanged during the execution of the protocol. By means of illustration, consider the exemplar protocol that involves the reviewer (as initiator of the protocol) and the review allocator in which the reviewer checks the correctness of the completed review form and sends it back to the review allocator.

Having completed the analysis phase, the first step of the design phase is to assign roles to agents (definition of the agent model). In this application, this is a trivial mapping for most of the roles since the agent model naturally derives from the real-world organisation: one PC Chair agent will assume both the roles of paper partitioner and review collector. One agent for each of the PC Members will assume the role of review allocator. Also, PC Member agents are likely to assume the role of reviewer, for those papers they review by themselves. Each external referee will be associated to an agent that will play the role of reviewer (for each paper that he is assigned).

From the agent model, the service model specifies which services the agent must implement to enable all the roles it has to play to be fulfilled. This amounts to transforming the abstract activities that the roles have to perform (as identified in the analysis phase) into more coherent blocks of computational activity. In the case of the reviewer role, for example, the only required service is the one that starts when the paper is received from the review allocator and then executes in order for the paper to be actually reviewed, and eventually sends back the completed review form.

Finally, the acquaintance model is constructed. This model, which simply identifies the communications pathways that exist between agents, provides a check of whether the structure of the interactions in the system are poorly organised. In this case, there are no obvious bottlenecks since the structure of the agent organisation closely resembles that of its real-world counterpart.

3.4 Limitations of the Gaia methodology

Gaia was intended for use in *closed systems* of *distributed problem solvers*, in which:

- the organisational structure of the system is static; i.e., neither the number of agents, nor their inter-agent relationships change at run-time;
- agents exhibit globally collaborative behaviour: they have a global goal in mind and do not exhibit competitive or self-interested behaviour.

However, these characteristics make Gaia, as it currently stands, unsuitable for Internet applications for the three main reasons that are dealt with in the remainder of this subsection. Each limitation is illustrated in the context of our conference management scenario.

3.4.1 No open systems

Gaia requires that all agents in a system, as well as all interaction protocols between them, are known a priori. However, there are cases, especially on the Internet, in which two agents need to interact with one another, although they do not know each other at design time and they do not have a preset interaction protocol for so doing.

In our application example, an external referee may want to rely on his own agent—well trusted and with a behaviour tuned to his own personal needs—for all its activities. Therefore, he may refuse to use an externally provided agent and, instead, insist that the PC member agent interacts directly with his personal agent (in order to receive the papers to review and to send them back to the PC Member agent). This type of situation is likely to become increasingly common as personal agents become more widespread.

In a system designed with Gaia, either the personal agent explicitly advertises itself and completely complies with the interaction protocols of the PC member agent or no interaction could take place. For example, Gaia would be unable to model a situation in which a PC Member agent has to contact an external referee agent to dynamically discover its area of competence.

3.4.2 No self-interested agents

Gaia does not explicitly deal with situations in which interactions are non-cooperative in nature. However, the Internet is populated predominantly by self-interested entities that compete to achieve their own objectives. It should be noted that Gaia does not preclude the development of such systems. Rather, it provides no explicit mechanisms for controlling or monitoring for such situations.

In the conference management example, if the system design does not account for self-interested behaviour then: *(i)* an author could make its own agent assume the role of reviewer for his own paper or *(ii)* a reviewer could obtain a significant number of papers to review and thus may exert a strong influence over the nature of the conference.

3.4.3 No Social Laws

More generally, perhaps the main drawback of Gaia is that it does not identify an organisation of agents as a *society* in which there are global laws that have to be respected by the society of agents.

As already stated, the definition of a multi-agent system has to rely not only on the individual task of each agents, but on the social task too and on the enforcement of the related social laws.

In the conference management example, a situation in which a reviewer receives too many (or too few) papers to review should be avoided. However, such situations can be difficult to monitor. On the one hand, an agent may be likely to exhibit self-interested behaviour and it may also lie in order to review either a lot of papers (to increase their influence in setting the scientific program) or a very few (to do less work). On the other hand, when PC Members autonomously allocate referees, one has to enforce the control of how many papers a single referee has received, possibly from different PC Members. The enforcement of this control requires specific interaction protocols that may impact in the definition of both PC Chair, PC Members, and Reviewer agents.

Gaia, by lacking the abstraction of a social task, does not explicitly model social laws and, as a consequence, either completely misses them or leaves them implicit in the definition of each agent.

4 Exploiting a Coordination Model

This section introduces the concept of a *coordination model* [15] and shows how it can be exploited in the context of designing multi-agent systems for use on the Internet. In particular, a coordination model makes it possible to enact social laws in the system and to control the execution of foreign and self-interested agents.

4.1 Coordination models

In general terms, coordination is the art of managing interactions and dependencies among activities, and a coordination model provides a formal framework in which the interaction of a set of concurrent activities can be expressed [15, 24].

In computer science, in its broadest conception, a coordination model deals with the creation and destruction, the communications, the distribution and mobility in space, as well as synchronisation over time [8], of a set of active software entities, whether processes, objects, or agents [29]. However, in most cases, a more restricted (and more manageable) view of a coordination model is adopted, as the framework dealing with the communication and synchronisation of a system of autonomous software agents.

From this latter perspective, a coordination model can be thought as consisting of three elements:

- the *coordinables*: the entities whose mutual interaction is ruled by the model, e.g., the agents in a multiagent system;
- the *coordination media*: the abstractions enabling agent interactions, as well as the core around which the components of a coordinated system are organised. Examples are semaphores, monitors, channels, or more complex media like tuple spaces, blackboards, etc.
- the *coordination laws*: define the behaviour of the coordination media in response to interaction events. The laws can be defined in terms of a *communication language* (a syntax used to express and exchange data structures) and a *coordination language* (a set of interaction primitives and their semantics)

According to [29], coordination models can be divided in two classes: *data-driven* and *control-driven*.

In *control-driven* coordination models, (e.g., Manifold [1]), coordinables (agents) interact with one another and with the external world via well-defined input/output ports, connecting the agents, and actually representing the coordination media. The observable behaviour of the coordinables, from the point of view of the coordination media, is in terms of state changes and events occurring on these ports. The coordination laws establish how events and state changes can occur and how they propagate though the coordination media. Therefore, the coordination media basically handle the interaction space by controlling how the flux of events connects the coordinables and how it propagates in the system, with no concern for the data exchanged between coordinables.

In *data-driven* coordination models, (e.g., Linda [14]), coordinables interact with the external world by exchanging data structures through the coordination media, which basically acts as a shared data space. The observable behaviour of the coordinables, from the point of view of the coordination media, is one of entities requesting data (either reading or extracting it from the dataspace). The coordination laws determine how data structures are represented and how they are stored, accessed, and consumed. Therefore, the coordination media basically handles the interaction space in terms of entities interacting via data exchange and synchronisation over data occurrences.

Although not all coordination models provide for a way of programming the coordination laws, which are then built into the coordination media, several recent proposals define flexible coordination media in which the default coordination laws can be changed and adapted to the specific coordination needs of a system [28, 6].

We refer the interested reader to Chapters 2 and 3, respectively, for a more detailed and formal characterisation of coordination models and for a survey on several control-driven and data-driven models. What is of interest here is that a coordination model, (whether data- or control- driven), models a system in terms of inter-agent interactions occurring occurring via coordination media embedding coordination laws, possibly programmable, to which all the interactions have consequently to conform. Adopting the above modelling in the analysis and design of multi-agent system provides a means of dealing with some of the shortcomings of the Gaia methodology.

Although both control-driven and data-driven models can be adopted to handle multi-agent system interactions, data-oriented models have an additional, important, advantage. In the Internet, in the majority of cases, agents exhibit an "information-oriented" view of their world (as argued in Subsection 2.4). Therefore, adopting a data-driven coordination model, which ensures all interactions occur via shared dataspaces, is a natural choice for such an information-oriented view of interaction.

The remainder of this section highlights how a coordination model perspective helps alleviate some of Gaia's aforementioned shortcomings.

4.2 Open Systems

When a coordination model is adopted, the coordination media mediates all interactions occurring between agents, and influences the effect of the interaction events according to its embedded coordination laws. As a consequence of this mediation, inter-agent interactions are intrinsically less coupled: for instance, a message sent by an agent crosses the coordination media before being delivered to another agent. If the coordination laws in the coordination media can programmed so as to embed any kind of behaviour or intelligence, the coordination media themselves can provide for a very loosely-coupled model of interactions. On the one hand, two agents can interact even if they do not know each other: the coordination media can take care of virtually connecting agents by appropriately delivering messages. On the other hand, two agents can interact even if they are heterogeneous in terms of their communication language or their interaction protocols: the coordination media can handle any necessary translation of messages and adaptation of protocols.

From the analysis and design point of view, the presence of the coordination media implies that it is no longer necessary to determine all the interaction links between all the possible agents that the application will meet—which a priori rules out the possibility of unforeseen agent arrivals—and to define all the possible interaction protocols. Instead, provided there is an appropriate design of the behaviour of the coordination media: *(i)* some of the interaction links between agents can be left unbound, by making the coordination media dynamically connect the interaction links to agents, according to its coordination law; *(ii)* agents do not need to be a priori aware of all possible interactions protocols, because the coordination can embed the functionality necessary to act as a "translator" in the interaction between heterogeneous agents.

In the conference management example, and with reference to the protocol involving the PC Member agent and the Reviewer agent in assigning a paper to review, the design should no longer consider that the PC Member agent will necessarily have an interaction link with a priori known reviewer agents. Instead, the design can leave the interaction link unbound, and the coordination media can be programmed so as to dynamically connect the link to the agent that will assume the role of reviewer. Then, the PC Member agent does not have to worry about whether the reviewer agents is a known reviewer agent or an unknown personal agent of the referee.

4.3 Self-interested agents

The coordination media, by mediating all interactions between agents, is intrinsically able to monitor and influence all interactions. Therefore, it can also be programmed in such a way as to constrain the behaviour of the agents during their interactions, for example by forbidding agents to send a specific kind of message to another agent. This can occur either by simply denying the access of the agents to the coordination media, or by modifying the semantics of the agents' interactions, for example by forwarding a forbidden message to a controller agent instead of delivering it to the original recipient.

This characteristic of the coordination media makes it comparatively easy to monitor and identify the presence of self-interested or opportunistic behaviours in the multi-agent system. Therefore, whenever these kinds of behaviour are likely to be damaging to the multi-agent system, the coordination media can be programmed so as to *(i)* recognise the appearance of such damaging behaviours and, thereafter, *(ii)* either forbid the access to the coordination media by the offending agent or simply constrain/modifiy the semantics of their accesses in order to make it harmless.

In the conference management example, a self-interested Reviewer agent that tries to collect too many papers to review, can be simply detected by the coordination media and can be excluded from further executing the interaction protocol needed to obtain papers.

4.4 Social laws

Since societies of any kind are grounded on the interaction among the individual components, it seems natural to identify the space of agent interactions as the place where social laws have to be represented and embodied. The interaction space has to be recognised as an independent design space for multi-agent system design, and abstractions and tools are needed to enable agent interaction to be shaped according to the social laws. If the interaction space is not recognised as a primary abstraction, (as is the case in Gaia), social laws have to be left implicit within the definition of roles and agents. However when the interaction space is recognised as the place where social tasks have to be modelled and social laws enforced, a coordination model is the appropriate conceptual abstraction around which to build a society of agents.

The adoption of a coordination model naturally leads to conceptualising a multi-agent system as a society, in which the individual tasks of agents have to be modelled separately from the social tasks. The definition of social tasks, in particular, instead of being mixed in the definition of roles and agents, find an appropriate model in the definition of the behaviour of the coordination media. In particular, by having the capability of monitoring and controlling all interactions between agents, the coordination media can take care of enforcing whatever social laws have to be respected by the agent system, in its entirity, in order for the social tasks to be met.

By considering the conference management example, we can make the concept of social laws clearer. The review process, to be effective and polite, has to be subject to several laws. For example: an author cannot act as a referee for their own paper; a referee should not be assigned to review two different papers of the same author; and the workload should be evenly distributed between the organising committee. All the above constraints cannot simply be reflected in the definition of a few roles and protocols. Instead, they are likely to impact the whole organisation of the conference. Therefore, once the social tasks are identified, the coordination media can become the place where the interactions between agents are appropriately monitored and controlled, so as to make agent interaction respect global goals.

5 Toward a Coordination-Oriented Methodology

The adoption of a coordination model introduces a further abstraction into the Gaia methodology that necessarily influences the way a multiagent system is analysed and designed (see Figure 3).

In the analysis phase, and with reference to the terminology adopted in Gaia, the characteristics of the multi-agent system have to be analysed and understood so as to identify not only the roles to be played in the system and their required interactions, but also the social laws to be respected by the multi-agent system. In other words, the analysis should clearly identify those aspects of the system that are spread across multiple roles and protocols, and therefore properly belong to the multi-agent systems as a whole, i.e., as a coherent organisation of agents with an expected global behaviour.

The expected output of the analysis phase are well-defined role and interaction models, in addition to a well-defined model of social laws. The latter specifies the social laws that must be respected and enforced by the system if it is to work properly.

In the design phase, the role model and the interactions model drive the design of individual agents and of their internal structure (i.e., with reference to the Gaia methodology, of the agent and service models). In addition, social laws, together with the identified interactions model, drive the definition of the actual expected behaviour of the coordination

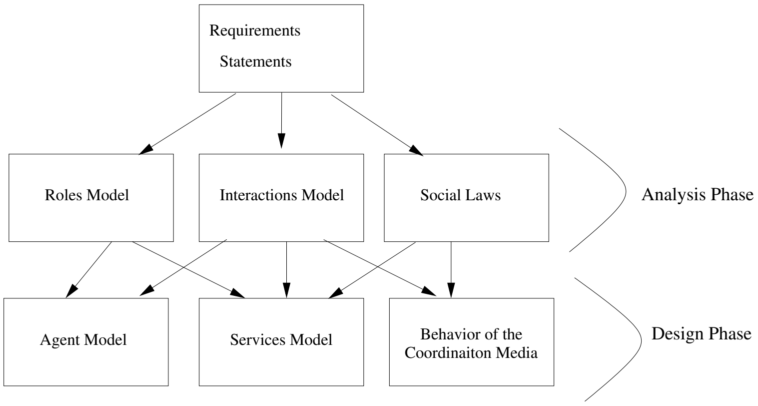

Figure 3: The basic concepts of a coordination-oriented methodology

media in response to the interaction events of the agents. Note that, because a methodology centered around a coordination model leads to an agent system intrinsically open and with dynamically bounded interaction links, the acquaintance model (defined by Gaia) is almost meaningless.

The expected output of the design phase should produce an agent and a service model detailed enough for the agents to be implemented (or designed making use of traditional object-oriented techniques). In addition, the behaviour of the coordination media should make it possible to implement the coordination media, whatever media it is actually exploited.

A coordination-oriented methodology obviously leads to a design specification of the coordination media that can very easily implemented by making use of a programmable coordination system, such as MARS [6], TuCSoN[28], and Law Governed Interactions [25], in which the concepts of mediated interactions via a programmable media have already found a clean and efficient implementation. Otherwise, should the multi-agent system rely on a traditional communication infrastructure (e.g., TCP/IP based message-passing), the actual implementation of the system should also include the implementation of the infrastructure needed to make all interactions compliant with the expected behaviour of the coordination media. For instance, this can be done by associating (even dynamically) specific communication stubs to the agents in the system, as was done, for example, in the FishMarket project [26].

6 Conclusions and Future Work

This paper has analysed the main issues that arise in engineering multi-agent systems for the Internet. Specifically it has shown how a coordination model can act as a powerful abstraction for both the analysis, design, and actual development of multi-agent systems.

An analysis of the state of the art in the area has shown that only a few complete and well-grounded methodologies for the analysis and design of multi-agent systems have been proposed so far. In particular, we focused on the Gaia methodology. However, as it currently stands, Gaia falls short when dealing with open agent systems. It does so because it lacks the necessary abstractions to deal with the dynamic arrival and departure of agents, and with self-interest agents. More generally, we have identified how Gaia does not support the concept of a multi-agent system as a society, in which agents have to act accordingly to specific social laws.

The adoption of a coordination model as the conceptual abstraction to be exploited in the analysis and design of multi-agent systems for the Internet, enables open systems, self-interested agents, and social laws, to find a suitable accommodation. On this basis, we have shown how the methodological concepts introduced by Gaia can be effectively complemented by the concepts of social laws and coordination media, leading to the definition of a coordination-oriented methodology suitable for multi-agent Internet systems.

In the latter part of this chapter, we identified the basic concepts and guidelines to be integrated in a methodology for the analysis and design of multi-agent systems on the Internet. Naturally, the sketched coordination-oriented methodology is far from being well-defined; it requires a more detailed specification, supported by appropriate formalism. In addition, there are several other issues—not specifically addressed by this chapter—that a software engineering methodology for multi-agent system on the Internet has to take into account. In particular:

- Object-oriented technologies have already recognised the need for re-use in object-oriented software architectures, and exploited them via design-patterns [13]. In the area of agent-oriented programming and multi-agent systems, we expect the same could apply with regard to the most widely used patterns of interactions between agents (see Chapter 14) and, even more generally, to the most widely used patterns after which a society of agent can be shaped. In other words, we expect agent-patterns to catalogue widely used social laws, to be exploited to re-use well-known behaviours of the coordination media.

- Interacting in open environments, such as the Internet, requires authentication mechanisms that allow agents to recognise each other, and control policies for enabling safe interactions (see Chapter 11). Therefore, analysis and design methodologies should provide appropriate abstractions to analyse security issues and define security policies. In this context, it is important to investigate whether a coordination model could be a suitable abstraction to define and handle security issues, by assuming the coordination media embed, in addition to embed the social laws, also the "security laws".

References

[1] F. Arbab, I. Herman, and P. Spilling. An overview of manifold and its implementation. *Concurrency: Practice and Experience*, 5(1):23–70, February 1993.

[2] G. Booch. *Object-oriented Analysis and Design (second edition).* Addison Wesley, Reading (MA), 1994.

[3] F. Brazier, B. Dunin-Keplicz, N.R. Jennings, and J. Treur. Formal specifications of multi-agent systems: a real-world case. In *First International Conference on Multi-agent Systems (ICMAS95)*, pages 25–32. June 1995.

[4] F.M.T. Brazier, B.M. Dunin-Keplicz, N.R. Jennings, and J. Treur. Desire: Modelling multi-agent systems in a compositional formal framework. *Journal of Cooperative Information Systems*, 6(1):67–94, 1997.

[5] B. Burmeister. Models and methodologies for agent-oriented analysis and design. In *Working Notes of the KI96 Workshop on Agent-oriented Programming and Distributed Systems*. DFKI, 1996.

[6] G. Cabri, L. Leonardi, and F. Zambonelli. Mars: a programmable coordination architecture for mobile agents. *IEEE Internet Computing*, to appear, 2000.

[7] P. Ciancarini, O. Nierstrasz, and R. Tolksdorf. A case study in coordination: Conference management on the internet. `http:www.cs.unibo.it/ cianca/wwwpages/case.ps.gz`.

[8] Paolo Ciancarini. Coordination models and languages as software integrators. *ACM Computing Surveys*, 28(2), June 1996.

[9] J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceeding of the 3rd International Conference on Multi-Agent Systems (ICMAS 98)*. IEEE CS Press, June 1998.

[10] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an agent communication language. In *Proceedings of the 3rd International Conference on Information and Knowledge Management*, Gaithersburg, Maryland, November 1994.

[11] Foundation for Intelligent Physical Agents. FIPA'99. `http://www.fipa.org`.

[12] M. S. Fox. An organizational view of distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):70–80, January 1981.

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison Wesley, Reading (MA), 1995.

[14] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.

[15] D. Gelernter and N. Carriero. Coordination languages and their significance. *Communications of the ACM*, 35(2):97–107, February 1992.

[16] N. Glase. *Contributions to Knowledge Modelling in a Multi-agent Framework (The CoMoMAS Approach)*. Ph.D Thesis, Université Henry Poincaré, Nancy (F), 1996.

[17] C. Iglesias, M. Garijo, J.C. Gonzales, and J.R. Velasco. Analysis and design of multiagent systems using mas-commonkads. In *Intelligent Agents IV (ATAL97), LNAI 1365*, pages 313–326. Springer-Verlag, 1998.

[18] Carlos Iglesias, Mercedes Garijo, and Juan Gonzales. A survey of agent-oriented methodologies. In A. S. Rao J.P. Muller, M. P. Singh, editor, *Intelligents Agents IV (ATAL98)*, LNAI. Springer-Verlag, 1999.

[19] N. Jennings. Agent-based computing: Promises and perils. In *International Joint Conference on Artificial Intelligence*. 1999.

[20] N. R. Jennings and M. Wooldridge. Agent-oriented software engineering. In *Handbook of Agent Technology*. ACM, 2000.

[21] N.R. Jennings, T.J. Norman, and P. Faratin. Adept: An agent-based approach to business process management. *ACM SIGMOD Record*, 27(4):32–39, 1998.

[22] E. A. Kendall. Role modelling for agent system analysis, design, and implementation. In *1st International Symposium on Agent Systems and Applications*. IEEE CS Press, October 1999.

[23] D. Kinny and M. Georgeff. A methodology and modelling technique for systems of bdi agents. In *Workshop on Modelling Autonomous Agents in a Multi-Agent World, LNAI 1038*, pages 56–71. Springer-Verlag, 1996.

[24] T. Malone and K. Crowstone. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119, 1994.

[25] N.H. Minsky, Y.M. Minsky, and V. Ungureanu. Making tuple spaces safer for heterogeneous distributed systems. In *Proceedings of the 2000 ACM Symposium on Applied Computing (SAC00)*. COMO, Italy, March 2000.

[26] P. Noriega. *Agent-mediated Auctions: The Fishmarket Metaphor*. Ph.D Thesis, Universitat Autonoma de Barcelona, Barcelona (E), 1997.

[27] J. Odell, H. Van Dyke Parunak, and C. Bock. Representing agent interaction protocols in uml. In *OMG Document ad/99-12-01*. Intellicorp Inc., December 1999.

[28] A. Omicini and F. Zambonelli. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, 1999.

[29] G.A. Papadopoulos and F. Arbab. Coordination models and languages. *Advances in Computers*, 46:The Engineering of Large Systems, August 1998.

[30] H.V.D. Parunak. Visulalizing agent conversations: Using enhanced dooley graphs for agent design and analysis. In *Second International Conference on Multi-agent Systems (ICMAS96)*, pages 275–282. June 1996.

[31] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[32] M. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):55–61, December 1998.

[33] Y. Tahara, A. Ohsuga, and S. Honiden. Agent system development based on agent patterns. In *International Conference on Software Engineering*, pages 356–367. ACM, 1999.

[34] M. Wooldridge and N. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[35] M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, to appear 2000.