# Developing Multiagent Systems: The Gaia Methodology

FRANCO ZAMBONELLI
Università di Modena e Reggio Emilia
NICHOLAS R. JENNINGS
University of Southampton
and
MICHAEL WOOLDRIDGE
University of Liverpool

Systems composed of interacting autonomous agents offer a promising software engineering approach for developing applications in complex domains. However, this *multiagent system* paradigm introduces a number of new abstractions and design/development issues when compared with more traditional approaches to software development. Accordingly, new analysis and design methodologies, as well as new tools, are needed to effectively engineer such systems. Against this background, the contribution of this article is twofold. First, we synthesize and clarify the key abstractions of agent-based computing as they pertain to agent-oriented software engineering. In particular, we argue that a multiagent system can naturally be viewed and architected as a *computational organization*, and we identify the appropriate organizational abstractions that are central to the analysis and design of such systems. Second, we detail and extend the Gaia methodology for the analysis and design of multiagent systems. Gaia exploits the aforementioned organizational abstractions to provide clear guidelines for the analysis and design of complex and open software systems. Two representative case studies are introduced to exemplify Gaia's concepts and to show its use and effectiveness in different types of multiagent system.

Categories and Subject Descriptors: D.2.10 [**Software Engineering**]: Design—*methodologies*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*multiagent systems*

General Terms: Algorithms, Design

Additional Key Words and Phrases: Multiagent systems, agent-oriented software engineering, analysis and design methodologies, distributed systems, software architectures

Authors' addresses: F. Zambonelli, Dipartimento di Scienze e Metodi dell'Ingegneria, Università di Modena e Reggio Emilia, Via Allegri 13–42100 Reggio Emilia, Italy; email: franco.zambonelli@unimo.it; N. R. Jennings, School of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, UK; email: nrj@ecs.soton.ac.uk; M. Wooldridge, Department of Computer Science, University of Liverpool, Liverpool L69 7ZF, UK; email: M.J.Wooldridge@csc.liv.ac.uk.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.
© 2003 ACM 1049-331X/03/0700-0317 $5.00

## 1. INTRODUCTION

The characteristics and expectations of software systems have changed dramatically in the past few years, with the result that a range of new software engineering challenges have arisen [Tennenhouse 2000; Zambonelli and Parunak 2003]. First, most software systems are now de facto concurrent and distributed, and are expected to interact with components and exploit services that are dynamically found in the network. Second, software systems are becoming “always-on” entities that cannot be stopped, restored, and maintained in the traditional way. Third, and as a natural consequence of the first two characteristics, software systems tend to be open, in that they exist in a dynamic operating environment where new components join and existing components leave on a continuous basis, and where the operating conditions themselves are likely to change in unpredictable ways. These characteristics apply, for example, to the semantic web [Berners-Lee et al. 2001], to grid computing [Foster and Kesselman 1999], and to pervasive environments [Abelson et al. 2000; Tennenhouse 2000].

Given this new landscape, we advocate the use of *multiagent systems* (MASs) as a software engineering paradigm for designing and developing complex software systems [Wooldridge and Jennings 1995; Jennings 2001; Wooldridge 2002]. In MASs, applications are designed and developed in terms of autonomous software entities (*agents*) that can flexibly achieve their objectives by interacting with one another in terms of high-level protocols and languages. These characteristics are well suited to tackling the emerging complexities for a number of reasons. First, the autonomy of the application components (i.e., the ability for an agent to decide what actions it should take at what time [Wooldridge and Jennings 1995]) reflects the decentralized nature of modern distributed systems [Tennenhouse 2000] and can be considered as the natural extension to the notions of modularity and encapsulation for systems that are owned by different stakeholders [Parunak 1997]. Second, the flexible way in which agents operate (balancing reactive behavior in response to the environment, with proactive behaviour towards the achievement of their design objectives [Wooldridge and Jennings 1995]) is suited to the dynamic and unpredictable situations in which software is now expected to operate [Zambonelli et al. 2001a]. Finally, the high-level and dynamic nature of multiagent interactions is appropriate to open systems in which the constituent components and their interaction patterns constantly change [Estrin et al. 2002; Ripeani et al. 2002; Ricci et al. 2002].

As is the case with any new software engineering paradigm, the successful and widespread deployment of complex software systems based on MASs requires not only new models and technologies, but also the identification of an appropriate set of software engineering *abstractions*. These abstractions serve as a reference for system identification and analysis, as well as providing the basis of *methodologies* that enable developers to engineer such systems in a robust, reliable, and repeatable fashion. In the last few years, there have been several attempts to identify appropriate abstractions for MASs, and to develop such software engineering methodologies accordingly. However, most of this

work is either tuned to specific systems and agent architectures (e.g., Ferber and Gutknecht [1998] and Bussmann [1998])—thus, it lacks generality—or else it is defined as a simple extension of existing object-oriented methodologies (e.g., Iglesias et al. [1999] and Bauer et al. [2001])—and is thus dependent on abstractions and tools that are unsuitable for modeling agent-based systems.

Against this background, a number of proposals have recently attempted to define complete and general methodologies, built around agent-specific abstractions and specifically tailored to the analysis and design of MASs [Wood et al. 2001; Caire et al. 2002; Bresciani et al. 2001; Wooldridge et al. 2000]. Although these methodologies adopt different terminologies and provide different abstractions, they all recognize that the process of building MASs is radically different from the process of building more traditional software systems. In particular, they all recognize (to varying extents) the idea that a MAS can be conceived in terms of an *organized society* of individuals in which each agent plays specific *roles* and interacts with other agents according to protocols determined by the roles of the involved agents (cf. the more traditional functional composition view of system architectures [Shaw et al. 1995]). However, in this article, we show that an organization is more than simply a collection of roles (as most methodologies assume), and that in order to effectively build a MAS in organizational terms, further organization-oriented abstractions need to be devised and placed in the context of a methodology.

To this end, this paper advances the state of the art in agent-oriented software engineering in two important ways:

- —It articulates the role of agent-based computing as a software engineering paradigm and identifies the set of *organizational abstractions* that are necessary for designing and building systems in complex, open environments.
- —It extends the Gaia methodology to exploit these new organizational abstractions. These extensions, while preserving the simplicity of the original Gaia proposal, enable it to be used in the analysis and design of open MASs (whereas previously it could only be used for closed communities of cooperating agents).

The remainder of this article is organized as follows: Section 2 details the key concepts of agent-based computing as they pertain to agent-oriented software engineering. It also outlines the similarities and differences with object- and component-based methods. Section 3 focuses on the organizational metaphor and describes and motivates the organizational abstractions that are necessary for agent-oriented software engineering. Section 4 details how these abstractions are exploited in Gaia in order to provide a methodology for the analysis and design of MASs. Section 5 discusses related work. Section 6 concludes and outlines some open issues.

## 2. A SOFTWARE ENGINEERING PERSPECTIVE ON AGENT-BASED COMPUTING

This section introduces the key concepts of agents and MASs and outline the main differences with work in object/component-based computing and software

architectures. The aim is to present the arguments as to why agent-oriented software engineering is a suitable paradigm for designing and building today's complex systems.

### 2.1 Basic Concepts

The first key concept is that of an *agent*; here viewed as a software entity exhibiting the following characteristics in pursuit of its design objectives [Wooldridge and Jennings 1995]:

—*Autonomy*. An agent is not passively subject to a global, external flow of control in its actions. That is, an agent has its own internal thread of execution, typically oriented to the achievement of a specific task, and it decides for itself what actions it should perform at what time.
—*Situatedness*. Agents perform their actions while situated in a particular environment. The environment may be a computational one (e.g., a Website) or a physical one (e.g., a manufacturing pipeline), and an agent can sense and effect some portions it.
—*Proactivity*. In order to accomplish its design objectives in a dynamic and unpredictable environment the agent may need to act to ensure that its set goals are achieved and that new goals are opportunistically pursued whenever appropriate.

For instance, a software component in charge of filtering emails can be viewed as a (simple) agent [Maes 1994]. It is autonomous if it is implemented as a threaded application logically detached from the client mail reader and if it is assigned the specific task of analyzing and modifying the content of some user mailboxes. It is proactive in that it can draw its user's attention to specific new mails or to specific situations occurring in its folders. It is situated in that it lives in a world of mailboxes and it can sense changes in the mailboxes and can effect their state.

Agents can be useful as stand-alone entities that are delegated particular tasks on behalf of a user (as in the above example and in goal-driven robots [Mataric 1992]). However, in the majority of cases, agents exist in environments that contain other agents. In these *multiagent systems*, the global behavior derives from the interaction among the constituent agents. This brings us to the second key concept of agent-based computing, that of *sociality* [Wooldridge and Jennings 1995]:

—agents interact (cooperate, coordinate or negotiate) with one another, either to achieve a common objective or because this is necessary for them to achieve their own objectives.

Broadly speaking, it is possible to distinguish between two main classes of MASs:

—*distributed problem solving systems* in which the component agents are explicitly designed to cooperatively achieve a given goal;

— *open systems* in which agents are not co-designed to share a common goal, and have been possibly developed by different people to achieve different objectives. Moreover, the composition of the system can dynamically vary as agents enter and leave the system.

The former class of systems includes applications that are primarily devoted to solving computationally intensive problems by assigning the task of exploring different portions of the search space to different agents, with the goal of expediting the search (e.g., Parunak [1997]). This class also includes those software systems that are devoted to controlling and directing a single physical process (e.g., Bussmann [1998]). Here, different agents are delegated the task of manipulating a sub-portion of the physical space in which the process takes place. In both cases, however, the system is closed; all agents are known a priori, they are supposed to be inately cooperative to each other and, therefore, they can trust one another during interactions. The latter class includes most distributed systems in which the agents have to make use of services, knowledge, and capabilities found in other agents spread throughout the network (e.g., agents for information retrieval [Cabri et al. 2000], for workflow management in virtual enterprises [Ricci et al. 2002], and for pervasive computing [Estrin et al. 2002]), as well as those systems that involve interactions between agents that represent different stakeholders (e.g., e-commerce agents [Esteva et al. 2001] and web service agents [Cabri et al. 2002]). In these cases, one must take into account the possibility of self-interested (competitive) behavior in the course of the interactions (preventing agents from inherently trusting each other) and the dynamic arrival of unknown agents.

### 2.2 Objects, Components, and Agents

The distinction between the contemporary view of objects and agents is becoming less sharp with time. However, let us first consider the "traditional" (historical) object-oriented perspective [Booch 1994]. An object, per se, is not autonomous, in that its internal activity can be solicited only by service requests coming from an external thread of control. As a consequence, an object is not capable of proactive behavior and is not capable of autonomously deciding what to do in a specific situation (i.e., an object is not capable of action selection). Moreover, in traditional object applications, there is not an explicit conception of "environment": objects either encapsulate resources in terms of internal attributes or perceive the world only in terms of other objects' names/references.

On the other hand, objects and components in today's distributed and concurrent systems are somewhat removed from this canonical definition and they are starting to approach our view of agents, at least in terms of observable behavior. Objects and components may be active and may integrate internal threads of execution enabling them to perform specific computational tasks. They may also have to serve requests in a concurrent way and have to preserve themselves from unauthorized or unsafe requests for services (in other words, they can select not to do an action). Finally, the coexistence in such systems of both active components/objects and passive objects, provides for a clear distinction between an active computational part of the system and a world of

external resources (i.e., of an environment). The fact that objects and components have to access such external resources and often have to deal with unexpected situations in so doing (exception and event handling) may require an explicit modeling of situatedness (such as the explicit identification of context-dependencies in component-based systems). Similar considerations can also apply to modern pervasive computer-based systems. When such entities enter into a complex interactive scenario (e.g., a wireless-enabled building for a PDA [Mamei et al. 2003], a wireless network of distributed sensors [Estrin et al. 2002] or a network of controllers in an automobile [Leer and Hefferman 2002]) their observable behavior, from the viewpoint of the system designer, is that of an autonomous agent.

In addition to the fact that modern object and component systems exhibit characteristics that make them assimilable to agents, agents themselves are often implemented in terms of active objects and components (e.g., Java threads with synchronization, exception handling, and event-handling capabilities). However, we emphasize the above mean neither that agents add nothing to modern objects or that the real advantages of agent-based computing can be appreciated only in the presence of AI techniques. Rather, the fact that objects and agents are converging from a technological viewpoint is evidence of the fact that, from a software engineering viewpoint, agent-based abstractions suit the development of complex software systems. This appears even clearer when shifting the attention from single object/components to complex systems.

### 2.3 Software Architectures and Multiagent Systems

Traditional object-based computing promotes a perspective of software components as "functional" or "service-oriented" entities that directly influences the way that software systems are architected. Usually, the global design relies on a rather static architecture that derives from the decomposition (and modularisation) of the functionalities and data required by the system to achieve its global goals and on the definition of their interdependencies [Bass et al. 2003; Shaw and Garlan 1996; Shaw et al. 1995]. In particular:

- —objects are usually considered as service providers, responsible for specific portions of data and in charge of providing services to other objects (the "contractual" model of software development explicitly promotes this view);
- —interactions between objects are usually an expression of inter-dependencies; two objects interact to access services and data that are not available locally;
- —everything in a system tends to be modeled in terms of objects, and any distinction between active actors and passive resources is typically neglected [Schwabe et al. 2002].

In other words, object-oriented development, while promoting encapsulation of data and functionality and a functional-oriented concept of interactions, tends to neglect modeling and encapsulation of execution control. Some sort of "global control" over the activity of the system is usually assumed (e.g., the presence of a single execution flow or of a limited set of controllable and globally synchronised execution flows). However, assuming and/or enforcing such

control may be not feasible in complex systems. Thus, rather than being at risk of losing control, a better solution would be to explicitly *delegate* control over the execution to the system components [Zambonelli and Parunak 2003]—as in MASs. In fact:

—Delegating control to autonomous components can be considered as an additional dimension of modularity and encapsulation. When entities can encapsulate control in addition to data and algorithms [Parunak 1997], they can better handle the dynamics of a complex environment (local contingencies can be handled locally by components) and can reduce their interdependencies (limiting the explicit transfer of execution activities). This leads to a sharper separation between the component-level (i.e., intra-agent) and system-level (i.e., inter-agent) design dimensions, in that also the control component is no longer global.
—The dynamics and openness of application scenarios can make it impossible to know a priori all potential interdependencies between components (e.g., what services are needed at a given point of the execution and with what other components to interact), as a functional-oriented perspective typically requires. Autonomous components delegated of their own control can be enriched with sophisticated social abilities, that is, the capability to make decisions about the scope and nature of their interactions at run-time and of initiating interactions in a flexible manner (e.g., by looking for and negotiating for service and data provision).
—For complex systems, a clear distinction between the active actors of the systems (autonomous and in charge of their own control) and the passive resources (passive objects without autonomous control) may provide a simplified modeling of the problem. In fact, the software components of an application often have a real-world counterpart that can be either active or passive and that, consequently, is better suited to being modeled in terms of both active entities (agents) and passive ones (environmental resources).

Again, we have to emphasize that modern approaches are beginning to recognize the above limitations. As outlined in Section 2.2, traditional object abstractions have been enriched by incorporating novel features—such as internal threads of execution, event-handling, exception handling, and context-dependencies—and are being substituted, in architectural styles, by the higher-level abstraction of self-contained (possibly active) coarse-grained entities (i.e., components). These changes fundamentally alter the way software architectures are built, in that active self-contained components intrinsically introduce multiple loci of control are more naturally considered as repositories of tasks, rather than simply of services. Also, the need to cope with openness and dynamics requires application components to interact in more flexible ways (e.g., by making use of external directory, lookup, and security services).

However, attempting to enrich more conventional approaches with novel features and characteristics to meet the novel needs, in addition to increasing the complexity of the modeling, is also likely to introduce a dangerous mismatch between the abstraction level adopted and the conceptual level at which

application problems have to be solved. Put simply, objects and components are too low a level of abstraction for dealing with the complexity of today's software systems, and miss important concepts such as autonomy, task-orientation, situatedness and flexible interactions. For instance, object- and component-based approaches have nothing to say on the subject of designing negotiation algorithms to govern interactions, and do not offer insights into how to maintain a balance between reactive and proactive behaviour in a complex and dynamic situations. This forces applications to be built by adopting a functionally oriented perspective and, in turn, this leads to either rather static software architectures or to the need for complex middleware support to handle the dynamics and flexible reconfiguration and to support negotiation for resources and tasks. Neither of these are particularly desirable.

In summary, we believe agent-based computing promotes an abstraction level that is suitable for modern scenarios and that is appropriate for building flexible, highly modular, and robust systems, whatever the technology adopted to actually build the agents.

## 3. THE ORGANIZATIONAL METAPHOR

Given the suitability of a modeling approach based on autonomous, situated agents that interact in flexible ways, there is a need to understand which further abstractions inspired by which metaphor complete the agent-oriented mindset.

### 3.1 Motivations

In recent years, researchers in the area of MASs have proposed a number of different approaches for modeling systems based on different metaphors, none of which can reasonably claim to be general purpose. For instance: the ant algorithms metaphor [Bonabeau et al. 1999; Babaoglu et al. 2002] has shown to be useful in efficiently solving complex distributed problems such as routing and distributed sorting; physical metaphors [Abelson et al. 2000; Mamei et al. 2003], focusing on the spontaneous reshaping of a system's structure, may have useful applications in pervasive and mobile computing; societal metaphors have been effectively applied in robotics applications [Moses and Tennenholtz 1995; Collinot et al. 1996] and in the understanding and control of highly-decentralized systems [Hattori et al. 1999; Ripeani et al. 2002].

Our approach focuses on the development of medium to large size systems, possibly dived in open and dynamic environments, and that have to guarantee predictable and reliable behaviors. For these kinds of systems, we believe the most appropriate metaphor is that of a *human organization* [Handy 1976; Fox 1981; Demazeau and Rocha Costa 1996; Zambonelli et al. 2001a], in which:

- A software system is conceived as the computational instantiation of a (possibly open) group of interacting and autonomous individuals (agents).
- Each agent can be seen as playing one or more specific roles: it has a well-defined set of responsibilities or subgoals in the context of the overall system and is responsible for pursuing these autonomously. Such subgoals may be

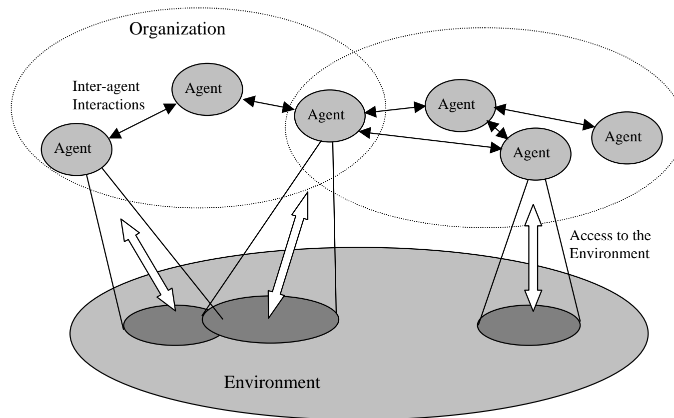


Fig. 1. Multiagent systems as computational organizations [Jennings 2001].

both altruistic (to contribute to a global application goal) or opportunistic (for an agent to pursue its own interests).

—Interactions are no longer merely an expression of interdependencies, and are rather seen as a means for an agent to accomplish its role in the system. Therefore, interactions are clearly identified and localized in the definition of the role itself, and they help characterize the overall structure of the organization and the position of the agent in it.

—The evolution of the activities in the organization, deriving from the autonomous execution of agents and from their interactions, determines the achievement of the application goal, whether an a priori identified global goal (as, e.g., in a workflow management systems where altruistic agents contribute to the achievement of a specific cooperative project), or a goal related to the satisfaction of individual goals (as, for example, in agent-mediated auctions, whose purpose is to satisfy the needs of buyer and seller agents), or both (as, for example, in network enterprises exploiting market mechanisms to improve efficiency).

The organizational perspective leads to a general architectural characterization of a MAS as depicted in Figure 1. Although some simpler systems can be viewed as a single organization, as soon as the complexity increases, modularity and encapsulation principles suggest dividing the system into different suborganizations (the dashed ellipses in Figure 1), with a subset of the agents being possibly involved in multiple organizations. In each organization, an agent can play one or more roles, to accomplish which agents typically need to *interact* with each other to exchange knowledge and coordinate their activities. These interactions occur according to patterns and protocols dictated by the nature of the role itself (i.e., they are institutionalized by the definition of the role). In addition, the MAS is typically immersed in an *environment* (i.e., an ensemble of resources, represented by the gray ellipse in Figure 1) that the agents may need to interact with to accomplish their role. Interactions with the environment occur via some sorts of *sensors* and *effectors* (i.e., mechanisms enabling

agents to perceive and act upon some part of the environment). That portion of the environment that agents can sense and effect (represented by the darker ellipses inside the environment in Figure 1) is determined by the agent's specific role, as well as by its current status.

The organizational metaphor—other than being a natural one for human developers who are continuously immersed in a variety of organizational settings and opening up the possibility of reusing a variety of studies and experiences related to real-world organizations [Handy 1976; Mintzberg 1979]—appears to be appropriate for a wide range of software systems. On the one hand, some systems are concerned with controlling and supporting the activities of some (possibly open) real-world organization (e.g., manufacturing control systems, workflow management and enterprise information systems, and electronic marketplaces). Therefore, an organization-based design may reduce the conceptual distance between the software system and the real-world system it has to support. On the other hand, other software systems, even if they are not associated with any pre-existing real-world organization, may have to deal with problems for which human organizations could act as fruitful source of inspiration, having already shown to produce effective solutions (e.g., resource sharing, task assignment, and service negotiation). More generally, whenever a software system is complex enough to warrant an agent-based approach and still requires a significant degree of predictability and reliability in all its parts, the organizational metaphor may be the most appropriate one. In fact, by relying on agents playing well-defined roles and interacting according to institutionalized patterns, the organizational metaphor promotes both micro-level (at the agents' level) and macro-level (at the system level) control over the design and understanding of the overall system behavior. Other metaphors (e.g., ant colonies and artificial societies) by focusing on the achievement of an average macro-level behavior of the system, often sacrifice the micro-level aspects (i.e., individual agents matter little or not at all). While this may be acceptable in, say, wide-area file sharing applications and heuristic allocation systems, it is definitely not in manufacturing control systems, enterprise information systems, electronics marketplaces, where each agent is important in its own right.

### 3.2 Example Applications

To illustrate our arguments, and to show how they map into real-world applications, we will consider two sample problems that will act as running examples throughout this article.

*Manufacturing Pipeline*. As an example of the class of distributed problem solving (closed) MASs, we consider a system for controlling a manufacturing process. Specifically, let us consider a manufacturing pipeline in which items are transformed or augmented (e.g., a pipeline in which metal items are painted).

Here, different agents may be devoted to the control of different stages of the pipeline (e.g., one agent is devoted to controlling the paint spraying, another to controlling the heat treatment of the paint, and yet another to controlling the cooling process). In such an organization, the role of each agent is that of "stage controller," in charge of ensuring that a specific portion of the pipeline works

properly (e.g., that the oven maintains a constant temperature and that the cooling system does not cool items too fast). To this end, agents need to sense and effect that portion of the environment which represents the stage of the pipeline of which they are in charge. Since portions of what an agent senses and effects may be sensed and effected by neighboring agents in the pipeline, some sort of indirect interaction, mediated via the environment, is likely to occur between agents. In addition, the agents may need to interact directly with each other to achieve a proper global functioning of the pipeline (for instance, by guaranteeing a uniform flux of items or by guaranteeing that the global flux of items does not exceed the processing capabilities of each of the stages).

*Conference Management.* As an example of a system raising issues typical of open systems—that is, agents exhibiting self-interested behavior, not all of whom may be known at deployment time—we consider an agent-based system for supporting the management of an international conference.

Setting up and running a conference is a multiphase process, involving several individuals and groups. During the submission phase, authors send papers, and are informed that their papers have been received and have been assigned a submission number. In the review phase, the program committee (PC) has to handle the review of the papers: contacting potential referees and asking them to review a number of the papers (possibly by bidding on papers). Eventually, reviews come in and are used to decide about the acceptance or rejection of the submissions. In the final phase, authors need to be notified of these decisions and, in case of acceptance, must be asked to produce a revised version of their papers. The publisher has to collect these final versions and print the proceedings.

The conference management problem naturally leads to a conception of the whole system as a number of different MAS organizations, one for each phase of the process. In each organization, the corresponding MAS can be viewed as being made up of agents being associated to the persons involved in the process (authors, PC chair, PC members, reviewers) to support their work, and representing the active part of the system. The roles played by each agent reflect the ones played by the associated person in the conference organization. This may require agents to interact both directly with each other and indirectly via an environment composed of (passive) papers and review forms.[1]

The openness of the conference management system is evidenced in a number of ways. Clearly, authors submitting a paper are unknown at deployment time, and so are the total number of author agents being involved in the process. However, this appears as a not very problematic issue, in that author agents (generally) have simply to send a paper to the PC chair, and do not need to strongly interact with other agents in the system. More problematic is the fact that, since agents are associated to different stakeholders with competing

[1]Such modeling differs considerably from traditional, object-based modeling of the same problem (see, e.g., Schwabe et al. [2002]), where the same application problem is typically modeled without making any distinction between the active entities and environmental resources (despite their being very different in nature) and where interactions are used to express functional interdependencies rather than the organizational relationships that actually cause them.

interests, unpredictable strategic and opportunistic behaviors can emerge in the application. The trivial case being that of a reviewer agent acting on behalf of a person which is also an author, and trying (e.g., by bidding for papers) to review its own paper or possibly competing papers. In addition, we also emphasize that a reviewer (or a PC member) may decide to exploit its own personal agent—rather than the agents made available by the conference management system—to enter the organization and interact with the other agents in it. In this case, even if all the reviewers are known in advance (and we know this is not the case in large conferences) the agents that will interact in the system may not be.

### 3.3 Organizational Abstractions

The characterization of a MAS of Figure 1 highlights the most basic abstractions that characterize a computational organization and that can be appropriately exploited in the analysis and design phases: the *environment* in which the MAS is immersed; the *roles* to be played by the different agents in the organization; and the *interactions* between these roles. In addition, we have identified two further abstractions that are often implicitly integrated into the above ones and that, we believe, need to be considered in their own right: *organizational rules* and *organizational structures*.

3.3.1 *The Environment.* A MAS is always situated in some environment and we believe this should be considered as a primary abstraction during the analysis and design phases. Generally speaking, identifying and modeling the environment involves determining all the entities and resources that the MAS can exploit, control or consume when it is working towards the achievement of the organizational goal. In some cases, the environment will be a physical one, including such things as the temperature in a room, the status of a washing machine or the average speed of cars on a motorway. This is the case in the manufacturing pipeline, where the agents are intended to control the correct processing and the global flux of the items in the pipeline. In other cases, the environment will be a virtual one, including such things as enterprise information systems, web services, and database management systems. This is the case of the conference management system, where agents execute in an environment populated by papers, review forms and digital libraries. Whatever the case, an explicit modeling of the environment is very important: not taking it into account (as, e.g., in the previous version of Gaia methodology [Wooldridge et al. 2000]) may complicate the overall design and may introduce mismatches between the MAS design and its actual operation. For instance, in the conference management example, the lack of environmental modeling would imply considering specific application agents as repositories of papers and review forms and as being in charge of explicitly transferring them to other agents. This is a more complex solution than that of explicitly modeling an environmental data repository, and conceptually farther from the likely operating environment (a world of Websites from which agents can access papers and review forms).

For both physical and computational environments, the following issues come to the fore when considering an explicit environmental modeling phase:

— *What are the environmental resources that agents can effectively sense and effect?* Constraints may exist with respect to the possibility of accessing and manipulating resources, and the environment model should therefore distinguish between the existence and the accessibility of a resource. In the manufacturing pipeline, it may be the case that no sensors exist to effectively evaluate the quality parameters. In conference management, it may be the case that a reviewer agent cannot pay to access a particular digital library (as needed to compare a submitted paper to an already published one). Also, since the environment may be characterised by its own dynamic, it may be the case that some of the resources are ephemeral in nature or that are only intermittently available. These dynamics may need to be identified, modeled, and evaluated against technological capabilities to cope with them.

— *How should the agent perceive the environment?* In other words, what representation of the environment is appropriate in a given situation? Such a choice is naturally dependent on the available technologies and on the pre-existing scenarios. In the manufacturing pipeline, the choice may depend on whether sensors are passive or active, leading to a characterization of the environment in terms of either a data world or a service world. In conference management, such a choice may depend on the characteristics of the available services providing access to papers and review forms.

— *What in the existing scenario should be characterized as part of the environment?* The existence of active entities with which the agents in the MAS will have to interact (e.g., computerbased sensors in the manufacturing pipeline or active databases in a Web scenario), means a decision has to be taken about what should be viewed as an agent and what should be viewed in terms of dynamic environmental resources. In other words, the distinction between the agent and the environment is not always clear cut. It is something that may require an accurate analysis and may ultimately depend on the problem's characteristics.

Summarizing, the environment of a MAS should not be implicitly assumed: its characteristics must be identified, modeled, and possibly shaped to meet application-specific purposes.

3.3.2 *Roles and Interactions.* The role of an agent defines what it is expected to do in the organization, both in concert with other agents and in respect of the organization itself. Often, an agent's role is simply defined in terms of the specific task that it has to accomplish in the context of the overall organization. However, our notion of a role is much more precise; it gives an agent a well-defined position in the organization, with an associated set of expected behaviors [Wooldridge et al. 2000; Ferber and Gutknecht 1998; Demazeau and Rocha Costa 1996]. Organizational role models precisely describe all the roles that constitute the computational organization. They do this in terms of their functionalities, activities, and responsibilities, as well as in terms of their interaction protocols and patterns. Organizational interaction models describe the

protocols that govern the interactions between the roles. Moreover, the interaction model describes the characteristics and dynamics of each protocol (e.g., when, how, and by whom a protocol has to be executed).

Most approaches to MAS modeling [Lind 2001; Wood et al. 2001], (including the first version of Gaia [Wooldridge et al. 2000]), consider role and interaction models as the sole organizational abstractions upon which to base the entire development process. Consequently, the analysis and design process starts directly with the identification and definition of role and interaction models. However, although role and interaction models can be useful to fully *describe* an existing organization, they are of limited value in *building* an organization. In fact, before the design process can define an actual organization (to be possibly described in terms of role and interaction models), there is a need to identify *how* the organization is expected to work and *which* kind of organization (among several possible ones) best fits the requirements identified in the analysis phase. This observation—elaborated below—motivates our introduction of the notions of organizational rules and organizational structures.

3.3.3 *Organizational Rules.* In the requirements capture phase of MAS development, it is certainly possible to identify the basic skills (functionalities and competences) required by the organization, as well as the basic interactions that are required for the exploitation of these skills. In the case of the manufacturing pipeline, it is easy to identify the need for agents to act as "stage controllers"; in the conference management case study, it is easy to recognize the need for agents to act as "authors," "reviewers," "PC Chair," and so on.

Although such an analysis can lead to the identification of a *preliminary* version of the role and interaction models, this identification cannot and *should not* be used to produce complete models. In fact, before being able to fully characterize the organization, the analysis of a MAS should identify the constraints that the actual organization, once defined, will have to respect. Typically, such constraints: (i) spread horizontally over all the roles and protocols (or, which is the same in this context, over the identified preliminary roles and protocols), or (ii) express relations and constraints between roles, protocols, or between roles and protocols [Esteva et al. 2001]. Although, in an actual organization, such constraints are likely to be somehow enacted by some agents playing some roles and interacting somehow, they can hardly be expressed in terms of individual roles or individual interaction protocols (in the same way, that social conventions and company directives horizontally influence our social life and our work, but cannot be associated with any specific actor). Thus, the explicit identification of such constraints—captured in our concept of *organizational rules*—is very important for the correct understanding of the characteristics that the organization-to-be must express and for the subsequent definition of the system structure by the designer.

The explicit identification of organizational rules is also important in the context of open systems. With the arrival of new, previously unknown, and possibly self-interested agents, the overall organization must be able to enforce its internal coherency despite the dynamic and untrustworthy environment. The identification of global organizational rules allows the system designer to

explicitly define: (i) whether and when to allow new agents to enter the organization, and, once accepted, what their position should be; and (ii) which behaviours should be considered as a legitimate expression of self-interest, and which among them must be prevented by the organization. In this context, organizational rules may also drive the designer towards the definition of the specific organization structure that most eases the enforcement of the organizational rules and, for instance, can facilitate the prevention of undesirable behavior on the part of the unknown agents.

In the manufacturing pipeline example, all the different stages must maintain the same speed of flow of items in the pipeline. This requirement is most simply expressed in terms of a global organizational rule, rather than replicating it as a requirement for each and every role in the organization (as, e.g., the previous version of Gaia and any methodology not making explicit use of organizational rules would have required). In the conference management system, there are a number of rules that drive the proper implementation of the organization. As notable examples: an agent should be prevented from playing both the role of author and reviewer of the same paper; PC members should not be in charge of collecting the reviews for their own papers. Neither of these constraints can easily be expressed in terms of properties or responsibilities associated to single roles or protocols and, if so, they would notably increase the complexity of roles and protocol description. Instead, they represent global organizational rules.

3.3.4 *Organizational Structures.* A role model describes all the roles of an organization and their positions in that organization. Therefore, a role model also implicitly defines the *topology* of the interaction patterns and the *control regime* of the organization's activities. That is, it implicitly defines the overall architecture of the MAS organization (i.e., its *organizational structure*). For example, a role model describing an organization in terms of a "master role" and "slave roles"—where the former is in charge of assigning work to the latter and of load balancing their activities—implicitly defines an organizational structure based on a hierarchical topology and on a load partitioning control regime. Other exemplar organizational structures include collectives of peers, multilevel and multidivisional hierarchies [Fox 1981], as well as more dynamic structures deriving from market-oriented models [Kolp et al. 2002]. All these organizations can be modeled in terms of a role model.

However, while the role model may define the organizational structure in an implicit way, the structure of a MAS is more appropriately derived from the explicit choice of an appropriate organizational structure, and organizational structures should be viewed as first-class abstractions in their own right. This argument (which is conventional in architecture-centered software design [Bass et al. 2003, Shaw and Garlan 1996]) calls for a specific design choice not to be (generally) anticipated to the analysis phase, but rather should exploit information collected during analysis. In the specific context of MAS development, the argument is motivated by several considerations:

—Although the organizational structure of a MAS may be directly inspired by the structure of the real-world system that the MAS must support,

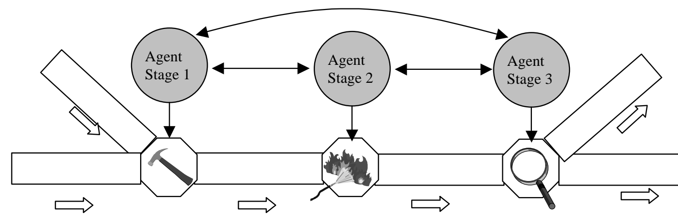


Fig. 2. Manufacturing pipeline: Collective of peers organization.

automate, or monitor (consider, for example, enterprise information systems and workflow management systems), this should not automatically imply that the organization of the software system should always mimic that of the real world system. This is so for several reasons. First, the real world organization may not be necessarily so well-structured. Second, the issues that may have driven a human organization towards the adoption of a particular structure may not necessarily apply to the agent organization. Third, the mere presence of software may introduce changes in the realworld organization. Of course, it may be the case that some specific sub-structures in a real-world organization are to be necessarily preserved at the MAS level and, thus, come predefined from the analysis phase. However, this should not be considered the general case and should not prevent developers from explicitly addressing the organizational structure issue.

—Starting from the organizational structure may prevent optimizing the overall efficiency of the organization and may prevent subsequent optimizations and changes. This consideration assumes a particular emphasis in dynamic applications scenarios (e.g., virtual enterprises [Ricci et al. 2002] global [Babaoglu et al. 2002] and pervasive computing [Estrin et al. 2002; Tennenhouse 2000]) where a system may need to frequently adapt its organizational structure to the prevailing situation, possibly at run-time and in an unsupervised way [Kephart and Chess 2003]. Although this paper does not explicitly deal with dynamic and unsupervised reorganizations, the methodological approach we propose (by making the definition of the organizational structure an explicit design decision) facilitates off-line reorganization and paves the way for supporting dynamic on-line reorganization.

—The organization, once defined, has to respect its organizational rules. Starting from a predefined organizational structure—by assuming to know in advance what it should be or by committing a priori to a given organizational structure—can make it difficult to have the organizational rules respected and enforced. It is more natural for the choice of the organizational structure to follow from the identification of the organizational rules.

In the manufacturing pipeline, perhaps the most natural choice is to have an organizational structure in which all of the stages in the pipeline form a collective of peers. For instance, with reference to Figure 2, the various stages of the pipeline are controlled by agents specifically devoted to controlling one stage, and each of these agents may directly interact with all the other stages to agree on issues requiring global coordination (e.g., regulating the flux of

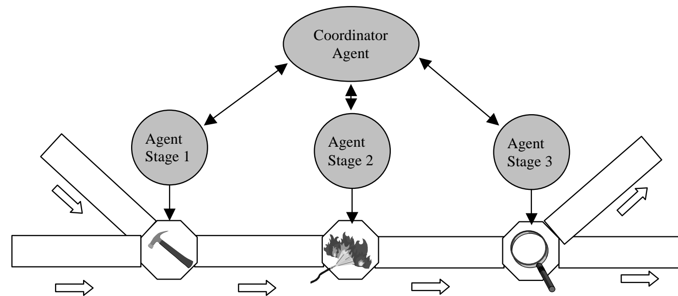


Fig. 3. Manufacturing pipeline: Hierarchical organization.

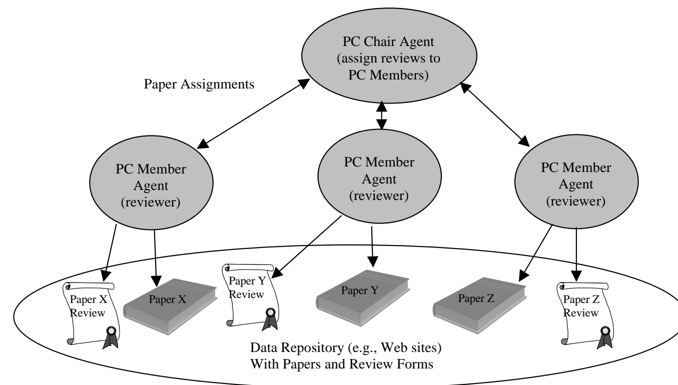


Fig. 4. Conference management: Single level hierarchy.

items in the pipeline). In other words, the topology of the interactions is a fully connected one and is subject to a fully distributed cooperative control regime. However, this is neither the only possible choice, nor necessarily the best one. For instance, because of the real-time nature of the pipeline control problem, it may be the case that some problem requiring complex global coordination between all the agents cannot be solved in time because of the high coordination costs associated with peer-to-peer interactions. In such cases, the designer can adopt a different organizational structure. For example, as sketched in Figure 3, a global coordinator agent can be introduced to control and mediate the interactions for all the other agents. This, in turn, leads to a hierarchical organization.

In the conference management, the overall structure of the MAS can generally be derived directly from the structure the conference officials have explicitly decided to adopt. A small conference usually relies solely on the PC members for the review process, with the PC chair acting as a global coordinator in a single-level hierarchy to control the reviewing work of the PC members (see Figure 4). The higher workload in a large conference usually requires a different approach (see Figure 5). For instance, the PC chair may partition the papers among the PC members, and the PC members may be in charge of finding reviewers for

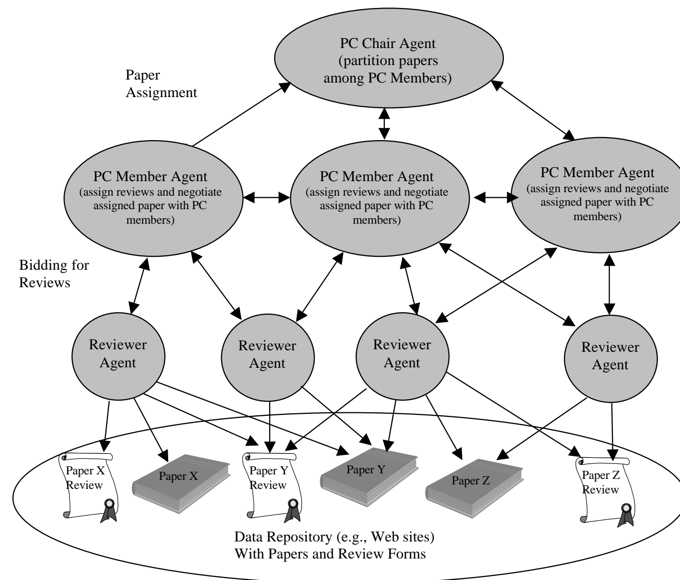


Fig. 5. Conference management: Multilevel hierarchy.

the papers in their partitions (either by asking a reviewer to review a specific paper or, alternatively, by calling for bids on papers in a sort of contract-net protocol [Smith 1980]). Also, PC members may negotiate with one another for re-allocating papers to different partitions. In other words, the organizational structure can be composed by a hierarchy at the higher level (ruling interactions between the PC Chair and the PC Members), and by a "market" organization at the lower level (to ensure a proper assignment of papers to reviewers and to enable direct negotiations between the collective of PC member agents). However, the development of a MAS for a conference must take into account the fact that the conference may change its dimensions (and its structure) from year to year and that the support of an agent-based system (reducing the work of the conference officials) may suggest a different organizational choice than that originally planned. Thus, if the analysis describes the system's requirements without committing to a specific organizational structure (as we suggest), the designer can reuse the analysis work to produce a new design according to the conference's new needs.

As an additional note related to organizational structures, we believe that despite the huge number of structures that can possibly be conceived, a (comparatively) small subset of these structures are likely to be used most of the time. This opens up significant opportunities both for re-use and for the exploitation of catalogues of agent-oriented *organizational patterns*—including use-cases reporting on efficiency, robustness, degree of openness, and ease of enactment of organizational structures—to support designers in choosing the most appropriate organizational structure for their problem. For instance, in the manufacturing example, the collective organization expresses a pattern that is likely to reappear in many applications. The same can also be said

of the hierarchical structure. In the conference management example, the various organizational structures that conferences of different sizes tend to adopt are also all fairly typical: single hierarchies, composite ones, contract nets, etc. Currently, in the area of agent-oriented software engineering, most pattern-related work focuses on detailed design patterns [Tahara et al. 1999; Kendall 2001]. In the near future, we expect more research and studies to be carried on in the area of architectural, organization-oriented patterns, to extend the specific studies already performed in this area regard [Fox 1981; Kolp et al. 2002] and to adapt those studies performed in the area of organization management [Mintzberg 1979; Handy 1976] for exploitation in agent-oriented methodologies.

## 4. THE GAIA METHODOLOGY

Having introduced and defined the various organizational abstractions that we believe are necessary for analyzing and designing MASs, the next step is to fashion them into a *design process*. That is, to produce an ordered sequence of steps, an identifiable set of models, and an indication of the interrelationships between the models, showing how and when to exploit which models and abstractions in the development of a MAS. The design process that we propose uses our previous work on the Gaia methodology [Wooldridge et al. 2000] as a point of departure. The new, extended version of Gaia (simply called Gaia hereafter) exploits the new organizational abstractions we identified in Section 3 and significantly extends the range of applications to which Gaia can be applied.

Before going into the details of Gaia, we first provide an overview of its key stages and models (see Figure 6). The Gaia process starts with the analysis phase, whose aim is to collect and organize the specification which is the basis for the design of the computational organization. This includes the identification of:

— *The goals of the organizations that constitute the overall system and their expected global behavior*. This involves identifying how to fruitfully decompose the global organization into loosely coupled suborganizations.
— *The environmental model*. Intended as an abstract, computational representation of the environment in which the MAS will be situated.
— *The preliminary roles model*. Identifying the basic skills required by the organization. This preliminary model contains only those roles, possibly not completely defined, that can be identified without committing to the imposition of a specific organizational structure. Also, the notion of roles, at this stage, is abstract from any mapping into agents.
— *The preliminary interaction model*. Identifying the basic interactions required to accomplish the preliminary roles. Again, this model must abstract away from the organizational structure and can be left incomplete.
— *The rules that the organization should respect and enforce in its global behavior*. Such rules express constraints on the execution activities of roles and protocols and are of primary importance in promoting efficiency in

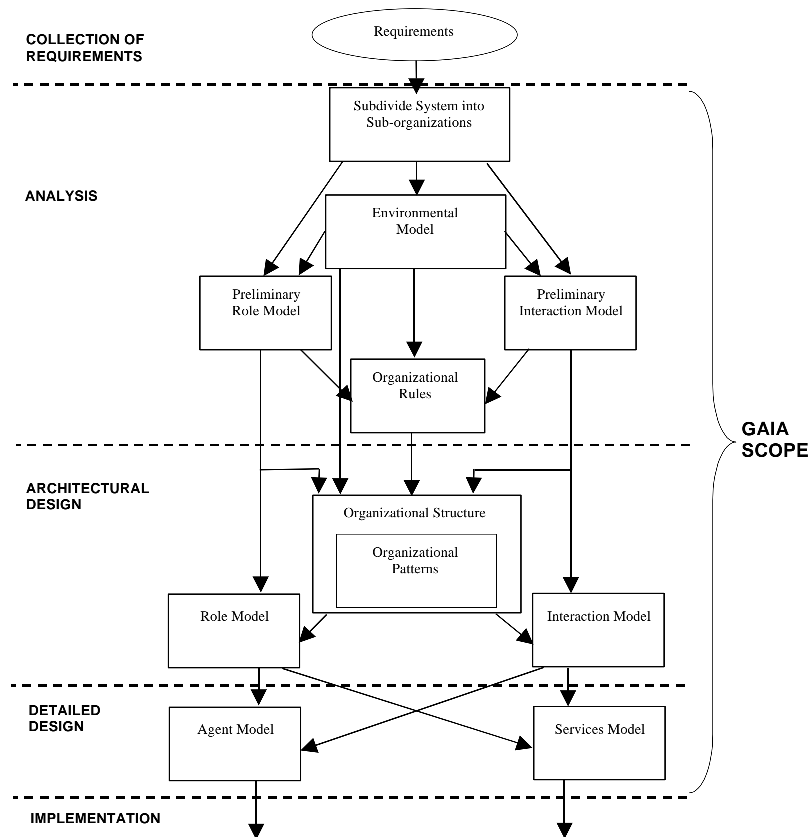


Fig. 6. Models of the Gaia methodology and their relations in the Gaia process.

design and in identifying how the developing MAS can support openness and self-interested behavior.

The output of the analysis phase—consisting of an environmental model, a preliminary roles model, a preliminary interactions model, and a set of organizational rules—is exploited by the design phase, which can be logically decomposed into an architectural design phase and a detailed design phase. The architectural design phase includes:

— *The definition of the system's organizational structure in terms of its topology and control regime*. This activity, which could also exploit of catalogues organizational patterns, involves considering: (i) the organizational efficiency, (ii) the real-world organization (if any) in which the MAS is situated, and (iii) the need to enforce the organizational rules.
— *The completion of the preliminary role and interaction models*. This is based upon the adopted organizational structure and involves separating—whenever possible—the organizational-independent aspects (detected from the analysis phase) and the organizational-dependent ones (derived from the adoption of a specific organizational structure). This demarcation promotes

a design-for-change perspective by separating the structure of the system (derived from a contingent choice) from its goals (derived from a general characterization).

Once the overall architecture of the system is identified together with its completed roles and interactions model, the detailed design phase can begin. This covers:

— *The definition of the agent model*. This identifies the *agent classes* that will make up the system and the *agent instances* that will be instantiated from these classes. There may be a one-to-one correspondence between roles and agent types, although a number of closely related roles can be mapped into in the same agent class for the purposes of convenience or efficiency.
— *The definition of the services model*. This identifies the main services—intended as coherent blocks of activity in which agents will engage—that are required to realize the agent's roles, and their properties.

Before detailing the analysis (Section 4.1), the architectural design phase (Section 4.2) and the detailed design phase (Section 4.3), it is important to explicitly state the scope and limitations of our methodology:

— *Gaia does not directly deal with particular modeling techniques*. It proposes, but does not commit to, specific techniques for modeling (e.g., roles, environment, interactions). At this time, and given the large amount of ongoing research work devoted to defining suitable (and possibly standardized) notation techniques for agent systems (as detailed in Section 5), we believe such a commitment would be premature.
— *Gaia does not directly deal with implementation issues*. The outcome of the Gaia process is a detailed but technology-neutral specification that should be easily implemented using an appropriate agent-programming framework (e.g., a FIPA-compliant agent system) or by using a modern object/component-based framework (i.e., one supporting distributed and concurrent objects). Of course, we are aware that specific technology platforms may introduce constraints over design decisions (a typical example relates to environmental modeling: if the environmental resources are of a passive nature, one cannot rely on having agents perceive them in terms of events). However, generally speaking, these situations should be known at the time of requirements capture.
— *Gaia does not explicitly deal with the activities of the requirements capturing and modeling, and specifically of early requirements engineering* [Mylopoulos et al. 1999]. We are aware of the importance of such activities and of the ongoing work in this area (see Section 5). In particular, we believe that Gaia could fit and be easily integrated with modern goal-oriented approaches to requirements engineering [Castro et al. 2002] whose abstractions closely match those of agent-oriented computing. However, the investigation of such an issue is beyond the scope of this article.

## 4.1 The Analysis Phase

The main goal of the analysis phase is to organize the collected specifications and requirements for the system-to-be into an environmental model, preliminary role and interaction models, and a set of organizational rules, for each of the (suborganizations) composing the overall system.

The basic assumption in Gaia is that the analysis phase can rely on the output produced by an early requirements engineering phase, devoted to analyzing the characteristics to be exhibited and the goals to be achieved by the system-to-be, as they emerge from the needs of the stakeholders and from the specific operational environment. The increasing acceptance of goal-oriented approaches to early requirements engineering [Mylopoulos et al. 1999]—modeling the specifications of systems in terms of the actors' involved, their roles, and their goals—is well suited to the organizational abstractions exploited by Gaia. Moreover, it may facilitate the Gaia analysis (which can accordingly be considered as a form of late requirements engineering phase), as well as the subsequent design phases.

4.1.1 *The Organizations.* The first phase in Gaia analysis is concerned with determining whether multiple organizations have to coexist in the system and become autonomous interacting MASs. Identifying such organizations is reasonably easy if (i) the system specification already identifies them or (ii) the system mimics the structure of the real world, and this involves multiple, interacting organizations. However, even if neither of these conditions hold, modularity concerns may suggest considering the system in terms of multiple organizations, for the sake of splitting the global complexity of a problem into a set of smaller more manageable components [Simon 1957]. Generally speaking, such suborganizations can be found when there are portions of the overall system that (i) exhibit a behavior specifically oriented towards the achievement of a given subgoal, (ii) interact loosely with other portions of the system, or (iii) require competences that are not needed in other parts of the system.

Turning to our running examples. The agents controlling the manufacturing pipeline are likely to be conceived as belonging to the same organization: (i) they share the same overall goal (making the pipeline run efficiently), (ii) they sometimes need to interact intensively with each other, while they interact less frequently with those agents concerned with controlling other parts of the factory (e.g., the electric plant), and (iii) the required agent skills are very closely related to that section of the manufacturing process. With regard to the conference management example, three suborganizations can be clearly identified. First, the organization responsible for the submission process is in charge of distributing the call for papers, collecting the submitted papers, and possibly doing some preliminary review/control of the papers. Second, the organization responsible for the review process. This is in charge of assigning submitted papers to reviewers, of collecting and ranking the reviews, and defining the technical programme. Third, the organization responsible for the publication of the proceedings. In all cases, there is a clear subgoal to be pursued in each organization, the interactions between organizations are loose and scheduled

at specific intervals of time, and there are entities that participate in some organizations and not in others.

4.1.2 *The Environmental Model.* It is difficult to provide general modeling abstractions and general modeling techniques because (as already stated in Section 3.1) the environments for different applications can be very different in nature and also becase they are somehow related to the underlying technology. To develop a reasonably general approach (without the ambition for it to be universal), we suggest treating the environment in terms of *abstract computational resources*, such as variables or tuples, made available to the agents for sensing (e.g., reading their values), for effecting (e.g., changing their values) or for consuming (e.g., extracting them from the environment).

Following such identification, the environmental model (in its simplest form) can be viewed as a list of resources; each associated with a symbolic name, characterized by the type of actions that the agents can perform on it, and possibly associated with additional textual comments and descriptions. A simple way to represent these is as follows (inspired by the FUSION notation for operation schemata [Coleman et al. 1994, pp. 26–31]):

reads *Var*1 // *readable resource of the environment*
*Var*2 // *another readable resource*
changes *Var*3 // *a variable that can be also changed by the agent*

Returning to our running examples. In the manufacturing pipeline, and assuming that the MAS is being deployed with the sole goal of ensuring the correct flow of items in the pipeline, the resources of interest are the number of items being produced and made flow in each of the pipeline's $n$ stages (other resources such as temperature, humidity, position, may be relevant for other specific activities of the stages). Thus, if appropriate sensors and actuators are available to detect and control, respectively, such flux, the environmental model reduces to:

changes $flux[i]$, $i = 1, n$ // *number of items flowing in each stage of the pipeline*

In the conference management, the resources handled by the agents are papers and review forms. In a minimal (exemplifying) representation, the environmental model can be as follows:

reads $papers[i]$, $i = 1, totalsubmitted$ // *all papers submitted for review*
changes $review[i][j]$, $i = 1, totalsubmitted;$
$j = 1, numberofreviewers$ // *reviews for the submitted papers*

Clearly, in realistic development scenarios, the analyst would choose to provide a more detailed and structured view of environmental resources. For instance, a paper would be represented by a data structure including information such as authors, title, keywords, and so on. Also, it is worth pointing out that more specific modeling techniques may be better adopted depending on specifications and technological constraints. For instance, the development of Web-based information based on XML documents (as may be the case in a conference

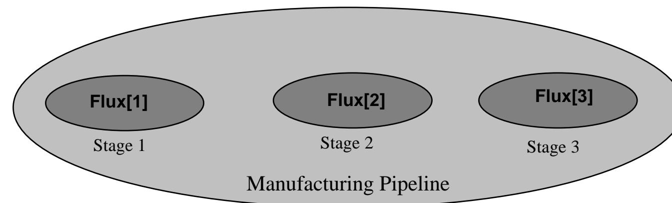


Fig. 7. Graphical representation of the manufacturing pipeline environment.

management system) might profit from an environmental model by preserving the already available XML data representation and simply enriching it with annotations related to how the MAS will access these structures.

In any case, there are other specific issues related to the modeling of the environmental resources that may be required to enrich/complement our proposed basic notation. In particular:

— Often, due to either the distributed nature of the environment or to the logical/physical relationships existing between its resources, a graphical (possibly annotated) scheme may be of help to represent such relationships and to identify how and from where a resource can be accessed. In the manufacturing pipeline, the various stages are intrinsically distributed in space, and it may be the case that the data related to the flux of an item in a particular stage can only be accessed by the stage itself. In this case, a graphical representation as per Figure 7 facilitates the capturing and understanding of the environmental characteristics. A similar representation can be used for the conference management environment where papers and review forms may be grouped either according to some logical relationships (e.g., the type of submission) or their network distribution (e.g., when a multiple track conference invites submissions directly to the Websites of the track chairs).
— The environment in which a MAS is immersed is typically characterized by its own dynamics. On the one hand, the content of the environmental resources may change over time according to patterns that may be of some relevance and may require some explicit modeling (for the sake of, for example, analytical modeling). These cases can generally be dealt with via annotations to the basic environmental model. On the other hand, some of the resources could be intermittently available, of an ephemeral nature, or simply be a priori unknown. These cases may also require enriching the basic access model to resources to take into account such uncertainties. For instance, an associative access model as per Linda tuple spaces may well suit this purpose [Cabri et al. 2000], as proved by its increasing adoption in open distributed systems [Cabri et al. 2002].

An important issue that should be borne in mind when modeling the environment is the fact that the operational environment of a MAS may include active components (i.e., services and computerbased systems) with which agents in the MAS have to interact. This introduces the problem of how to deal with such entities: should they be modeled as resources of the environment, or should they instead be "agentified"? (i.e., modeled in terms of additional agents in the

MAS) Some general guidelines can be provided with respect to this issue:

— When the role of these active components is simply that of a data provider (consider, for example, a Web server or a DBMS mediating access to a dataset, or a simple computerbased sensor), it is better to abstract away from their presence and to model them in terms of resources. The rationale for this is that their presence influences only the mechanisms by which agents retrieve resources (i.e., obtaining the data by requesting a service rather than by performing a sensing operation), not the nature of the resources themselves or the internal activities of the agents. Similar considerations may apply for simple components capable of event-notification services (i.e., upon change of a resource value).
— If the environment contains components and services that are capable of performing complex operations (e.g., active databases, active control systems, humans in-the-loop) then their effects on the agents' perception of the environment can make it hard to model them as a simple resource repository with identifiable patterns of dynamic change to be sensed by agents (or to interact with based on event-notification mechanisms). In such cases, these components should not be treated as part of the environment but, instead, they should be agentified.

4.1.3 *The Preliminary Role Model.* The analysis phase is not intended to design the actual organization of the MAS (this is the purpose of the subsequent architectural design phase). However, even without knowing what the structure of the organization will be, it is possible, in most cases, to identify some characteristics of the system that are likely to remain the same independently of the actual organizational structure. In particular, this equates to identifying the "basic skills" that are required by the organization to achieve its goals, as well as the basic interactions that are required for the exploitation of these skills. Such identification activities may even be facilitated if a goal-oriented early requirements analysis has already modeled the characteristics of the system in terms of actors involved and their goals.

Given the identification of the basic skills and of their basic interaction needs, respectively, the analysis phase can provide a *preliminary definition* of the organization's roles and protocols. However, this definition cannot be completed at this stage. In fact, the basic skills (or *preliminary roles*) can only truly become organizational roles when it is known how and with which other roles they will interact. This, in turn, requires the definition of the global organizational structure. Analogously, the basic interaction needs (or *preliminary protocols*) can only be fully defined as organizational protocols when the adoption of an organizational structure clarifies which roles these protocols will involve, when their execution will be triggered, and by whom. In addition, the design phase is likely to introduce additional roles and protocols, directly derived from the adopted organizational structure and, therefore, not identified in the analysis phase.

As anticipated in Section 3.3.3, it may also be the case that the need to adopt specific organizational substructures in a system derives from specific

requirements, in which case at least some of the roles and protocols may assume an already complete definition even at this stage.

In the manufacturing pipeline, the preliminary roles (PR) that can be devised are those associated with each of the stages in the pipeline:

$$\text{PR} = \text{Stage}[1], \text{Stage}[2], \ldots, \text{Stage}[N].$$

In addition, depending on the global structure of the manufacturing process, it is possible to devise other preliminary roles. For example, when the flux of items in the pipeline require more power than is currently supplied, an additional role may have to be introduced whose task is to coordinate with the electric plant (better, the agent organization of the electric plant) and negotiate further supplies of energy. However, unless the design has identified whether the overall system should be structured as an organization of peers (Figure 2) or a hierarchy (Figure 3) the role specification cannot be completed, nor is it possible to identify all the roles involved in the organization. Analogously, it may be clear from the analysis phase that a protocol should be defined for enabling one stage to ask for a slowing down in the flux of items in the pipeline. However, in this case, it is not certain which other entity this protocol should involve (the other stages in the pipeline or a global controller, depending on the chosen organizational structure).

In the conference management case, and in particular in the organization of the review process, it is comparatively easy to devise a number of preliminary roles that will be played whatever the organizational structure. For example, the roles associated with selecting reviewers and assigning papers to them (ReviewCatcher), the role of reviewer in charge of filling review forms for assigned papers (Reviewer), the role in charge of collecting and ranking the reviews (ReviewCollector) and the role of finalizing the technical programme (DoProgram). Depending on the actual organizational structures, different actors (e.g., the PC Chair, the PC Members or external reviewers) will be called to play such roles. However, the actual organizational structure, defined in the architectural design phase, may also require additional roles to be introduced. For example, a big conference may opt to subdivide the submitted papers among a set of co-chairs according to their competences. In this case, an additional role is required, for partitioning the submitted papers according to the defined criterion. This is likely to influence the definition of the protocols and to require the definition of further protocols (e.g., protocols to achieve some form of load balancing on the partitions).

To represent (preliminary) roles, Gaia adopts an abstract, semiformal, description to express their capabilities and expected behaviors. These are represented by two main attribute classes, respectively: (i) permissions and (ii) responsibilities.

*Permissions.* These attributes are mainly aimed at: (i) identifying the resources that can legitimately be used to carry out the role—intuitively, they say what *can* be spent while carrying out the role; and (ii) stating the resource limits within which the role must operate—intuitively, they say what *can't* be spent while carrying out the role. In general, permissions relate agent roles

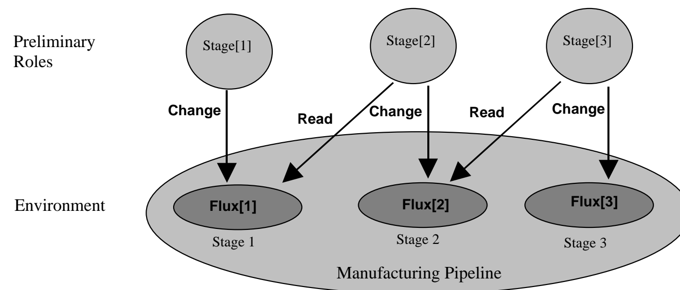


Fig. 8. The environment-preliminary roles diagram.

to the environment in which they are situated: in order to carry out a role, an agent will typically have to access environmental resources and possibly change or consume them. However, they can also be used to represent *knowledge* the agent can possibly have or have received from communications with other roles. To represent permissions, Gaia makes use of the same notation already used for representing the environmental resources. However, the attributes associated with resources no longer represent what can be done with such resources (i.e., reading, writing, or consuming) from the environmental perspective, but what agents playing the role must be allowed to do to accomplish the role and what they must not be allowed to do.

In the manufacturing pipeline example, an agent in charge of controlling a specific stage of the pipeline may need to sense (read) the abstract resource representing the flux of items entering its region of responsibility. However, since the previous stage is controlled by a different agent, it cannot control (change) it. Nevertheless, the role requires an agent to sense and effect the flux of items in its stage, representing the input flux to the next stage. This reduces to the following representation of the permissions for a generic role STAGE[$i$]:

| | | |
|---|---|---|
| reads | $flux[i-1]$ | // *flux from previous stage* |
| changes | $flux[i]$ | // *flux to next stage* |

In the conference management example, the REVIEWER role requires the capability to read the papers it has been assigned for review and of writing the associated review forms:

| | | |
|---|---|---|
| reads | *Papers* | // *all the papers it receives* |
| changes | *ReviewForms* | // *one for each of the papers* |

Due to the strict relations between the permission and the environmental models, a graphical representation relating the preliminary roles and their connections with the environment is often a useful outcome of the analysis phase. To this end, Figure 8 details such a graphical representation for the manufacturing pipeline example. Specifically, such an environment-role diagram may help identify inconsistencies between what operations the environment permits and what the agents need (or must be allowed) to do. The relations between the environment and the roles' permissions (identified in Figure 6 by the arrow going from the environmental model box to the preliminary role model box)

helps identify several key points about the developing MAS:

—Suppose an environmental resource is available for reading only, whereas one of the identified roles needs to change it. In this case, either the environmental model has to be rethought (if possible) or the capabilities of the roles have to be reduced or re-distributed among roles.
—In physically distributed environments, the diagram helps to determine whether it is reasonable and feasible for a single role to access a wide variety of widely distributed resources, or whether it is more appropriate to divide these capabilities among a set of roles.
—When a role needs access to a variable to which it does not have direct access, a possible solution is to access it via the mediation of a role that does. This may help in identifying the need for a given role to be involved in a specific protocol to gain mediated access to that variable. This inter-dependency of the environmental model and of the preliminary interaction model is denoted in Figure 6 via a proper arrow.

*Responsibilities*. These attributes determine the expected behavior of a role and, as such, are perhaps the key attribute associated with a role. Responsibilities are divided into two types: *liveness properties* and *safety properties* [Manna and Pnueli 1995]. Liveness properties intuitively state that "something good happens," that is, describe those states of affairs that an agent must bring about, given certain conditions. In the conference management example, a liveness property for the REVIEWER role may specify that a reviewer, on receipt of a paper, has to start the review and eventually produce a completed review. In the manufacturing pipeline example, a liveness property may specify that whenever a substantial change occurs in the flux of items, the STAGE controllers must engage in some coordinated activity to restore a stable condition. In contrast, safety properties are *invariants*. Intuitively, a safety property states that "nothing bad happens," that is, that an acceptable state of affairs is maintained. In the manufacturing pipeline, the role of STAGE controller must guarantee that the flux of items is always kept within a specific range.

The most widely used formalism for specifying liveness and safety properties is temporal logic, and the use of such a formalism has been strongly advocated for use in agent systems [Wooldridge 2000]. Although it has undoubted strengths as a mathematical tool for expressing liveness and safety properties, there is always some doubt about the viability of such formal tools for use in everyday software engineering practices. Therefore, without excluding the possibility of using such a formalism, this article proposes an alternative approach based on regular expressions, as these are likely to be understood by a larger audience. To this end, liveness properties are specified via a *liveness expression* which defines the "life-cycle" of the role (see Table I). Liveness expressions are similar to the *life-cycle* expressions of FUSION [Coleman et al. 1994], which are in turn essentially regular expressions. Our liveness expressions have an additional operator, "$\omega$," for *indefinite repetitions*.

Liveness expressions, defining the potential execution trajectories through the various activities and interactions associated with the role, have the general

Table I. Operators for Liveness Expressions

| Operator | Interpretation |
|---|---|
| $x.y$ | $x$ followed by $y$ |
| $x \mid y$ | $x$ or $y$ occurs |
| $x*$ | $x$ occurs 0 or more times |
| $x+$ | $x$ occurs 1 or more times |
| $x^{\omega}$ | $x$ occurs indefinitely often |
| $[x]$ | $x$ is optional |
| $x \parallel y$ | $x$ and $y$ interleaved |

following form:

$$\text{RoleName} = \textit{expression}$$

where RoleName is the name of the role whose liveness properties are being defined, and *expression* defines the liveness properties of RoleName. The atomic components of a liveness expression are either *activities* or *protocols*. An activity is somewhat like a method in object-oriented terms, and corresponds to a unit of action that the agent performs and that does not involve interaction with any other agent. Protocols, on the other hand, are activities that *do* require interaction with others. To give the reader some visual clues, we write protocol names in a sans serif font, and use the same font, underlined, for activity names.

To illustrate liveness expressions, let us consider the responsibilities of a simple Reviewer role in the conference management example:

$$\text{Reviewer} = (\mathsf{ReceivePaper}.\underline{\mathsf{ReviewPaper}}.\mathsf{SendReviewForm})^{maximum-number}$$

This expression says that Reviewer consists of executing the protocol ReceivePaper (for the moment, we treat the protocols simply as labels—we will give a more accurate definition shortly), followed by the activity ReviewPaper and the protocol SendReviewForm. The sequential execution of these protocols and activities is then repeated for the maximum number of papers the role has to deal with. In any case, we emphasize that during the analysis phase it may be impossible to completely determine the liveness expression for a role. For instance, without being committed to a specific organizational structure, one cannot determine if the reception of a paper by a reviewer has to be preceded by some sort of negotiation or bidding. In such cases, the preliminary roles definition can only sketch the activity (i.e., the liveness expression) of a role, to be completed during design.

As an additional example, consider the liveness expression for a role Stage$[i]$ in the manufacturing pipeline example:

$$\text{Stage}[i] = (\underline{\mathsf{MonitorFlux}}[i-1]^{\omega}.\mathsf{ReduceSpeed}.\mathsf{OKReduceSpeed})^{\omega} \parallel \underline{\mathsf{ProcessItems}}^{\omega}$$

This expression says that Stage$[i]$ consists of continuously executing an indefinite number of times the activity MonitorFlux$[i-1]$, to continuously check the flux of incoming items, and to be possibly sometimes interrupted and followed by the execution of the protocols ReduceSpeed and OKReduceSpeed.

These two protocols, intended to negotiate a new speed and actualize the speed change in the pipeline, respectively, have to be executed whenever the safety rule expressing the fact that the input and output fluxes must be the same cannot be respected. In parallel with the above continuous activities, is the—again continuous—ProcessItems activity, devoted to controlling item processing in the stage.

We now turn to safety requirements. These can be specified by means of a list of predicates, typically expressed over the variables listed in a role's permissions attribute. Returning to our REVIEWER role, an agent carrying out this role will be required to ensure that for any received paper it has also received the review form. This can be expressed by the following safety expression:

—$number_of_papers = number_of_review_forms$

By convention, we simply list safety expressions as a bulleted list, each item in the list expressing an individual safety responsibility. It is assumed that these responsibilities apply across *all* states of the system execution. Thus, if the role is of infinitely long duration, the invariants should always be true. However, because something unexpected can always occur in complex systems we prefer to assume a weaker notion for safety expressions; considering them as something that the agent playing a role has to do its best to preserve. In other words, safety expressions indirectly express abnormal situations to which an agent should be able to react. Of course, the need to respect safety expressions may require an agent to tune its liveness properties accordingly. This requirement appears clearly in the manufacturing pipeline example. Here a basic safety rule for the role STAGE[$i$] it to ensure that the flux of items exceeds for no more than a given threshold $T_i$ (and possibly for no more than a short time interval) that it is able to process:

—$flux[i] = flux[i-1] \pm T_i$

To ensure such a property, STAGE[$i$] must monitor the flux and eventually request a flux reduction, as already identified when discussing the liveness expressions for this role.

With all these definitions in place, it is now possible to precisely define the Gaia roles model. This is used in the analysis phase to define preliminary roles and in the design phase to give the complete specification of all the roles involved in a system. A roles model is comprised of a set of *role schema*, one for each role in the system. A role schema draws together the various attributes discussed above into a single place (Figure 9). In the case of the preliminary role model, such a schema may simply leave some of its parts undefined, to be completed in the architectural design phase. An exemplar instantiation is given for the REVIEWER role in Figure 10.

4.1.4 *The Preliminary Interaction Model.* This model captures the dependencies and relationships between the various roles in the MAS organization, in terms of one protocol definition for each type of inter role interaction. Since the roles model is still preliminary at this stage, the corresponding protocols model must also necessarily be preliminary, for the same reasons.

| Role Schema: | *name of role* |
|---|---|
| Description | *short description of the role* |
| Protocols and Activities | *protocols and activities in which the role plays a part* |
| Permissions | *"rights" associated with the role* |
| Responsibilities | |
| Liveness | *liveness responsibilities* |
| Safety | *safety responsibilities* |

Fig. 9. Template for role schemata.

| Role Schema: REVIEWER |
|---|
| Description: This preliminary role involves receiving papers for review from some conference official, reviewing that paper, and sending back a completed review form. |
| Protocols and Activities: ReceivePaper, <u>ReviewPaper</u>, SendReviewForm |
| Permissions: reads *Papers* // *all the papers it receives*; changes *ReviewForms* // *one for each of the papers* |
| Responsibilities<br>Liveness: REVIEWER = (ReceivePaper.<u>ReviewPaper</u>.SendReview)$^{maximum_number}$<br>Safety: • $number_of_papers = number_of_reviewforms$ |

Fig. 10. Schema for role REVIEWER.

In Gaia, a protocol can be viewed as an institutionalized pattern of interaction. That is, a pattern of interaction that has been formally defined and abstracted away from any particular sequence of execution steps, to focus attention on the essential nature and purpose of the interaction, rather than on the precise ordering of particular message exchanges (cf. the interaction diagrams of OBJECTORY [Coleman et al. 1994, pp. 198–203] or the scenarios of FUSION [Coleman et al. 1994]).

A protocol definition consists of the following attributes:

— *Protocol Name*. brief textual description capturing the nature of the interaction (e.g., "information request", "schedule activity X" and "assign task Y");
— *Initiator*. the role(s) responsible for starting the interaction;
— *Partner*. the responder role(s) with which the initiator interacts;
— *Inputs*. information used by the role initiator while enacting the protocol;
— *Outputs*. information supplied by the protocol responder during interaction;
— *Description*. textual description explaining the purpose of the protocol and the processing activities implied in its execution.

In the manufacturing pipeline, consider the ReduceSpeed protocol, which forms part of all STAGE[*i*] roles (Figure 11). This protocol is initiated by one of the stages in the pipeline when it notices that the input flux of items can no

| Protocol Name: **Reduce Speed** | | |
|---|---|---|
| Initiator: **Stage [i]** | Partner: **??** (Stage[I-1] OR Controller) | Input: **proposed new speed** |
| Description: When a stage cannot afford the current speed of items, it has to start a protocol to negotiate a new speed. | | Output: **new speed** |

Fig. 11. The ReduceSpeed preliminary protocol definition.

| Protocol Name: **Receive Paper** | | |
|---|---|---|
| Initiator: **??** (PC Chair or PC Member) | Partner: **Reviewer** | Input: **Paper info** |
| Description: When a paper has to be assigned to a reviewer it (by someone undefined at this stage) it will be proposed by sending paper info to one of the potential reviewer | | Output: **No, don't review** **OR** **Yes, I review it, send me the full paper** |

Fig. 12. The ReceivePaper preliminary protocol definition.

longer be tolerated and that a global reduction in pipeline flux is required. At this stage of the development process, however, it is not possible to know which other roles this protocol will involve. In any case, it is possible to identify that the outcome of the protocol will be a new value, properly negotiated, for the flux of items. In the conference management case study, consider the ReceivePaper protocol which is part of the REVIEWER role (Figure 12). This states that the protocol ReceivePaper will be initiated by some role (still undefined at this stage) in charge of assigning papers for review. It involves the role REVIEWER being asked if it can review the proposed paper, eventually leading to REVIEWER receiving the *Paper* (i.e., the specific paper for which the protocol was initiated).

Of course, a more detailed protocol definition (including, e.g., the sequencing of messages within the protocol) can be added at any time to complement our rather conceptual definition and provide more specific guidelines for design and implementation. For instance, the AUML notation could serve this purpose well [Odell et al. 2001].

4.1.5 *The Organizational Rules.* The preliminary roles and interaction models capture the basic characteristics, functionalities, and interaction patterns that the MAS system must realize, independently of any predefined organizational structure. However, as previously stated, there may be general relationships between roles, between protocols, and between roles and protocols that are best captured by organizational rules. In Gaia, the perspective on

organizational rules is consistent with that on roles' responsibilities: organizational rules are considered as *responsibilities* of the organization as a whole. Accordingly, it is possible to distinguish between safety and liveness organizational rules. The former refer to the invariants that must be respected by the organization for it to work coherently, and the latter express the dynamics of the organization (i.e., how the execution must evolve). In particular:

— *liveness* rules define how the dynamics of the organization should evolve over time. These can include, for example, the fact that a role can played by an entity only after it has played a given previous role, or that a given protocol may execute only after some other protocol. In addition, liveness organizational rules can relate to other liveness expressions belonging to different roles, that is, relating the way different roles can play specific activities.

— *safety* rules define time-independent global invariants for the organization that must be respected. These can include, for example, the fact that a given role must be played by only one entity during the organization's lifetime or that two roles can never be played by the same entity. In addition, they can relate to other safety rules of different roles or to expressions of the environmental variables in different roles.

Due to their similar nature, organizational rules can be expressed by making use of the same formalism adopted for specifying liveness and safety rules for roles.[2] To illustrate this, consider the following organizational rules, expressing exemplar *liveness* rules:

— $R \rightarrow Q$. Means that the role $Q$ can be played by an entity only if it has somewhen earlier played the role $R$.

— $R^3 \rightarrow Q$. Means that the role $Q$ can be played by an entity after it has somewhen earlier played at least three times the role $R$.

Conversely, safety expressions for organizational rules detail properties that must always be true during the whole life of the MAS:

— $\neg(R \mid Q)$. Means that the two roles $R$ and $Q$ can never be played concurrently by the same entity (recall that the "|" operator for liveness rules expresses concurrency of activities).

— $\neg((R|0)|(Q|0))$. Means that the two roles $R$ and $Q$ can never be played alone, that is, that an agent must play the two roles concurrently;

— $R^{1\ldots N}$. Means that the role $R$ must always be played at least once and no more than $N$ times;

— $R(property)$, $R = Role1; Role2; RoleN$. Means that the specified *property* has to be spread over all the roles listed.

In the manufacturing pipeline, the correct management of the pipeline requires each of the stage roles to be played only once. This can be expressed by

---

[2]The interested reader can refer to Zambonelli et al. [2001b] for a treatment of organizational rules that makes use of temporal logic and leads to a more formal approach. Again this is not incorporated here for reasons of ease of use by practitioners.

the safety rule:

$$R^1,\ R = (\textsc{Stage}[1], \textsc{Stage}[2], \ldots, \textsc{Stage}[N])$$

As a liveness organizational rule for this example, it is possible to consider the fact that, in addition to the tolerated bias between the flux of items in a single stage, the whole pipeline could be required to have a maximum tolerated bias among all the stages. Such a property, to be intended as a global responsibility of the whole MAS, can be expressed by the following organizational rule:

$$\mathit{flux}[1] = \mathit{flux}[2], \ldots, \mathit{flux}[N], \pm T_{global}$$

Turning to the conference management example. The rule expressing the fact that an author cannot act as a reviewer for its own paper can be expressed as:

$$\neg(\textsc{Reviewer}(\mathit{paper}(x)) \mid \textsc{Author}(\mathit{paper}(x)))$$

The rule that the role of reviewer must be played at least three times for each of the submitted papers (i.e., each must receive at least three reviews) is expressed as:

$$\textsc{Reviewer}(\mathit{paper}(i))^{3+},\ i = 1, \ldots, \mathit{number_of_submitted_papers}$$

(Such a rule should be coupled with another one avoiding a referee to review a paper multiple times.) The rule expressing the fact that the role in charge of setting up the conference program can only be played after that in charge of collecting all the reviews has been played is expressed as:

$$\textsc{ReviewCollector} \rightarrow \textsc{DoProgram}.$$

Similarly, liveness and safety organizational rules can be imposed on protocols. For example:

— $P^1$. The protocol must be executed only once;
— $P(R_1)^1$. The protocol must be executed only once by role $R_1$;
— $P \rightarrow Q$. The protocol $P$ must necessarily precede the execution of protocol $Q$;
— $P(R1) \rightarrow Q(R_1)$. The protocol $P$ must necessarily be executed by role $R_1$ before $R_1$ can execute protocol $Q$;

In the manufacturing pipeline, one of the stages may need to request a reduction in the speed of items because it can no longer operate at the current speed. However, this reduction of the speed (requested by the protocol ReduceSpeed) can only be actualized (e.g., via a protocol OKReduceSpeed) after the protocol ReduceSpeed has involved all the other stages. This can be expressed as:

$$\text{ReqReduceSpeed}^{N-1} \rightarrow \text{OKReduceSpeed}.$$

In the conference management example, the rule expressing the fact that the reviewer of *paper*($x$) must send one and only one review for that paper, by executing the protocol SendReview, is:

$$\text{SendReview(Reviewer(paper(x)))}^1.$$

The rule expressing the fact that only after a reviewer has received a paper for review can it actually send back the review form (by initiating the appropriate protocol) is:

$$\text{ReceivePaper(Reviewer(paper(x))))} \rightarrow \text{SendReview(Reviewer(paper(x)))}.$$

The above two rules, together with those previously introduced for expressing the fact that each paper must receive three reviews, ensure that each paper will receive three reviews from different referees. In summary, organizational rules, and their correct identification, are fundamental to the design phase because they constrain the definition of the actual organizational structure population and restrict the number of proper implementations of an organization. In addition, the definition of organizational rules is necessary when the system is open. In this case, organizational rules are important to analyze (i) whether and when a new agent can be admitted into the organization and which roles it is allowed to play and (ii) which kinds of protocols can be initiated by the members of the organization and when, and which kinds of protocols are an expression of legitimate selfinterest. As a final note, the above examples of organizational rules clearly show that collecting and defining the system specifications may also imply discovering which agents have to be present in the organization. Some of the organizational rules for roles are likely to be reflected in the design phase, driving, for example, the number of agents to be instantiated and the number of roles to be played by a single agent. However, although some of these design decisions may become clear well before the design phase, the analysis phase should still disregard the presence of agents and only focus on the more abstract concept of roles.

### 4.2 The Design Phase: Architectural Design

The output of the Gaia analysis phase systematically documents all the functional (and to some extent nonfunctional) characteristics that the MAS has to express, together with the characteristics of the operational environment in which the MAS will be situated. These structured specifications have to be used in architectural design to identify an efficient and reliable way to structure the MAS organization, and to complete accordingly the preliminary roles and interactions models.

It is worth emphasizing that, while the analysis phase is mainly aimed at understanding what the MAS will have to be, the design phase is where *decisions* have to be taken about the actual characteristics of the MAS. Therefore, even if some of the activities in design (i.e., the completion of the roles and interactions models) appear to be aimed at refining the outputs of the analysis, they in fact rely on actual decisions about the organizational structure and lead to a modeling of the MAS actual characteristics starting from the specifications.

Of course, any real world project faces the difficult task of identifying when the analysis phase can be considered complete (i.e., mature enough for design decisions to be taken). In most cases, only the initiation of design activities enables missing or incomplete specifications to be identified, or conflicting requirements to be noted. In either case, such findings typically require a

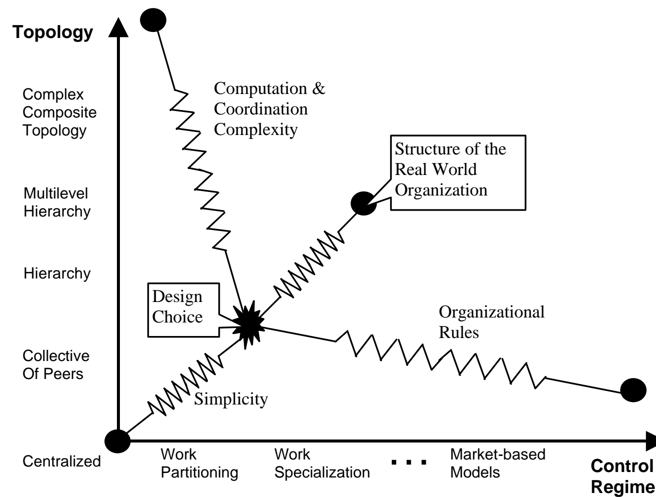


Fig. 13. The different forces involved in the identification of the organizational structure.

regression back to earlier stages of the development process. Gaia is not exempt from such problems, although by making explicit some decisions which are implicit in other methodologies (i.e., the architectural design), it promotes an earlier identification of them.

4.2.1 *Choosing the Organizational Structure.* The choice of the organizational structure is a very critical phase in MAS development, affecting all subsequent phases. Unfortunately, as is always the case with architectural design, it is not possible to identify a precise and formal methodology with which to obtain the "best" design. Nevertheless, a design methodology could and should give guidelines to help the designer make choices.

As Figure 13 illustrates, there are a number of (nearly orthogonal) forces that may drive the identification of an appropriate organizational structure. These include the choice of an appropriate *topology* and of an appropriate *control regime*, among a range of possible choices for the two dimensions. The forces affecting this choice may include: the need to achieve organizational efficiency (or, equivalently, the need for the MAS to properly handle the computation and coordination complexity of a problem); the need to respect organizational rules; and the need to minimize the distance from the real-world organization. All of which may be in tension with the desire to keep the design simple.

*Organizational Efficiency and Simplicity.* Organizational theory, which has been successfully applied in the analysis and design of distributed software systems [Fox 1981], states that any member of an organization, whether a human being, a hardware component, or a software agent, exhibits *bounded rationality* [Simon 1957]. That is, the amount of information it is able to store and process in a given amount of time is limited. Therefore, for an organization to work efficiently it must not overload its members. Whenever the problem of bounded rationality is likely to affect the organization, the organization has to

enlarge its resources (i.e., of its members), so as to better distribute the workload among them. The fact that organizations of different sizes may require different organizational topologies derives from the coordination costs that are incurred as the number of members in the organization increases.

As a general guideline (see Figure 13), the simplest organization *topology* that makes it possible to properly handle the computational and coordination complexity should be chosen. At one extreme, a single-member organization experiences no coordination costs and has the advantage of extreme simplicity, although in most cases it is impossible to charge a single agent with all the duties. When an organization is composed of only a few members, the coordination costs can be sufficiently low that collective decisions among all the members are possible. The resulting organizational network is a *collective of peers* in which all members, although possibly committed to different roles, have the same authority over the global organization. When the size of the organization increases still further, the members of the collective can no longer bear the increased coordination costs and a hierarchical topology must be adopted. In the hierarchy, the organization's members are freed from the duty of handling the coordination activities because a *leader* member assumes responsibility for them. As the organization's size increases further, a collective leadership may be needed to handle the increased coordination costs, or a multilevel hierarchy may emerge. In general, any composition of the basic structures can be adopted for the definition of the overall organizational structure. Also, in some cases, hybrid organizational networks can emerge. For example, a collective of peers can be ruled by a leader for a limited portion of its global coordination needs, while maintaining the capability of direct peer-to-peer coordination for others (or, equivalently, a leader in a hierarchy can take charge of a limited set of coordination activities, leaving subordinate members to directly coordinate with one-another for the remaining ones).

The *control regime* handling the interactions between the members of the organization is somewhat orthogonal to the organizational topology. However, in contrast to our simplified representation in Figure 13, it may also somehow influence the organizational efficiency (although in a less significant and direct way than the topology does). In a collective of peers, for example, the control regime can be based either on a *workload partitioning* regime (in which each of the peers has the same role and provides the same services) or on a *workload specialization* (in which each of the peers provides a specific service/activity). The two solutions, depending on the specific application characteristics, may lead to different coordination costs, thus influencing organizational efficiency: when large numbers of similar tasks have to be performed, work partitioning is better at reducing complexity and coordination costs; when the overall workload derives from a few complex activities, work specialization may incur lower coordination costs. Similar considerations may apply to different control regimes applied to different topologies. For instance, it is possible to opt for a hierarchy in which the main goal of the leader is to coordinate the activities of the subordinates so as to obtain a sensible partitioning (i.e., load balancing) of work among them. Alternatively, it is possible to opt for a hierarchy in which each subordinate is specialized, to supply either a specific service or product,

and in which the leader is in charge of the allocation of resources and services so that each subordinate can work at its full load. In the case of very large organizations and when competitive and self-interested behavior are likely to emerge, both the cooperative control regimes of collectives and the authoritative control regimes of hierarchies are often ineffective. This is because of the unavoidable inefficiencies of multilevel hierarchies (or of hierarchies with high fan-out). In such cases, a control regime based on market models for the distribution of workloads and the supply of services needs to be adopted.

In the manufacturing pipeline example, we have already identified (Section 3.3) that both a collection of peers (each in charge of a specific piece of work and directly coordinating with each other) and a hierarchy (in which the leader handles all the coordination needs of the stages) are possible. The choice between these alternatives must consider whether the coordination costs between stages can be directly sustained by the stage roles or not. In the conference management example, we have already discussed how the number of papers to be dealt with in the review process—to be sustained by humans of bounded rationality—influences the structure of the real-world organization and, consequently, of the supporting agent-based systems.

As an additional note, it is worth pointing out that additional forces could influence the choice of the organizational structure and, specifically, its topology. For instance, a problem which is intrinsically distributed in physical space (as in the manufacturing pipeline) and in which the resources of the environment cannot be accessed by every agent from everywhere at the same costs, may mandate the adoption of certain specific distributed topologies.

*The Influence of Organizational Rules*. Another factor impacting on the choice of the organizational structure (primarily of the control regime, as from Figure 13, although not exclusively of it) is the need for the MAS to respect organizational rules and to be able to enact them during execution.

Some types of organizational rules, typically safety ones, may express very direct and intuitive constraints directly driving the adoption of specific organizational structures (e.g., constraints specifying that two roles have to be played by the same agent or that the same agent cannot play two specific roles together). Clearly, in these cases, and whatever the specific topology adopted, the control regime must be tuned so as to distribute tasks in the organization in respect of the constraints. In the conference management example, the fact that a submitted paper has to be reviewed by at least three different reviewers simply rules out the possibility, during review assignments, to consider a work partitioning regime for the sake of load balancing.

The most significant case in which organizational rules impact on the organizational structure is in the presence of competitive and self-interested behavior. In such cases, not only is it necessary that the adopted structure of a system respects the organizational rules, but also that the adopted structure promotes their active enactment. In fact, on the one hand, agents that dynamically enter an organization may be unaware of the local rules, and, on the other hand, self-interested agents may strive to opportunistically ignore such rules to achieve their own specific aims. In such cases, the organizational structure

and specifically the control regime of interactions must be shaped so as to make it possible to enact some form of control over the execution of foreign and self-interested agents (or, equivalently, to control which agents can play a specific role and how they play it) so that the organizational rules are respected overall. In the case of a big conference, for example, the need for the PC Members to evenly and efficiently distribute papers to reviewers suggests the adoption of a market-inspired bidding mechanism to rule the activities at the lowest level of the multilevel hierarchy. This solution, potentially the most efficient one to implement review assignments, would rely on making the list of papers public and on having papers assigned to those reviewers expressing their availability and willingness to review them. However, to avoid a reviewer handling its own papers (and, more generally, not to leave a reviewer a free choice about which papers to review) the control regime cannot be based solely on this mechanism. Rather, the bidding process has to be considered only as a preliminary phase, to guide the final and authoritative decisions of the PC Members about review assignments.

The above discussion also highlights the fact that the enforcement of some organizational rules may have a significant effect on the computational or coordination costs (i.e., in the above example, the need to control review assignment charges PC Members of the additional tasks of supervising the bidding process). Thus, organizational rules may impact on the choice of the organization topology. For instance, consider a liveness rule that forces a specific ordering on the execution of a set of interaction protocols involving a group of agents. In this case, the adoption of a simple collective of peers as a basic topology would impose so much load on each of the agents to control the enactment of the rule that a designer would do better to opt for a hierarchy, with a leader in charge of controlling the protocols' execution sequence. A situation of this kind may also occur in the manufacturing pipeline. Here there is a need to ensure that all stages have executed the OKReduceSpeed protocol before the ReduceSpeed one can be executed. This may, in turn, be the primary force suggesting the adoption of a hierarchical topology instead of a collective one.

*The Influence of the Real-World Organization.* In many cases, a MAS is intended to support the structure of a real-world organization. Therefore, in general, the structure of the real-world organization acts as an attracting force in the definition of the organizational structure (as shown in Figure 13), with a natural tendency to define it so as to mimic/reflect the real-world structure. Notable examples are agent-based systems for computer supported cooperative work, in which agents are associated with each of the team members and in which agents usually have to cooperate according to the interaction patterns of the real-world organization. In a similar way, in agent-based systems for electronic commerce, the patterns of interactions between the agents and their organizational roles are likely to mimic the ones that can be found in human commercial transactions.

Having the structure of the MAS reflect the structure of the real-world organization may be important for the sake of conceptual simplicity, and in several cases this may also come as a requirement, at least for some portions of the

structure. However, is not always the case that this attracting force should be accommodated.

Since the construction of a MAS necessarily implies an in-depth analysis of the corresponding real-world organization, it often happens that this process uncovers inefficiencies in the real-world organization. These inefficiencies may simply be related to the fact that the organizational structure is not appropriate for the size and objectives of the organization, or it may be more subtle. The latter case includes the discovery of role underloads (e.g. the duties of a member of the organization are significantly lower than those it could cope with) or role ambiguities (e.g., some members of the organization do not have a well-defined role or they misunderstand the responsibilities). In such cases, and when the designer of the system does not have the authority to modify and improve the real-world organization, it is still possible to improve its software counterpart.

In addition, the availability of software systems to support the work of teams and organizations tends to strongly impact on the way of working and on the real-world organizational structure itself, even if it were previously efficient (a phenomenon that applies to all information technology placed in this situation). Once the members of the organization can have part of the work delegated to agents, their role in the organization can become underloaded, and the whole real-world organization may be forced to rethink its structure to face the reduced computational and coordination costs in a more efficient way. Where possible, this problem has to be taken into account in the design of the software system. However, since the problem is most likely to show its effect over time, after the MAS has entered its life in the real-world organization, the designer may be forced to tune the chosen organizational structure of the MAS to these new needs.

The conference management example exemplifies the above influences well. For instance, the PC Chair in charge of "manually" assigning papers to reviewer PC Members (i.e., for each and every paper, by finding at least three PC Members to act as reviewer for that paper, as per Figure 4) may be overwhelmed by this work as the number of papers becomes large. Therefore, as is usual in large conferences, the PC Chair is forced to simply partition the papers among PC Members (as per Figure 5) and have the PC Members help themselves to find the needed number of reviewers for their assigned papers (i.e., delegating to PC Members the role ReviewCatcher). However, when an agent-based system is provided to support the conference organization committee, it may be the case that the work of the PC Chair and of the PC Members dramatically reduces and, for instance, the PC Chair may be able to directly recruit reviewer PC Members for a much larger number of papers.

4.2.2 *Exploiting Organizational Patterns.* Whatever the specific factors that may determine the choice of an organizational structure, it is likely that similar issues have been faced in the past, and have led to similar considerations and choices. This is the stage where the availability of a catalogue of possibly modular and composable "organizational structures," describing in much more detail than we have done how and when to exploit a specific structure, will greatly help the designer [Shaw and Garlan 1996; Kolp et al. 2002; Castro

et al. 2002] and will effectively complement our general guidelines. In fact, the guidelines we have described are intended to be of help in evaluating the right trade-off in a rather coarse-grained taxonomy of structures and control regimes. However, they consider neither a number of variations on the theme that could be conceived, nor a number of nonfunctional aspects possibly affecting such a more detailed choice (e.g., mobility, security, resilience, etc.). The specific identification of these variations and their impact on specific non-functional aspects may be well supported by a structured catalogue of organizational structures. This will help the designer to reuse both well-documented and motivated design choices and the design work related to the representation of such a structure.

4.2.3 *Representing the Organizational Structure.* Once the identification of a suitable organizational structure for the MAS is complete, the designer has to determine how to effectively represent it. In this context, we suggest the coupled adoption of a formal notation and of a more intuitive graphical representation.

The obvious means by which to formally specify an organization is to explicate the inter-role relationships that exist within it (to represent the topology of the organizational structure) and their types (to represent the control regime of the organization). We emphasize that there is no universally accepted ontology of organizational relationships. Nevertheless, as a first pass towards a more complete characterization and formalization, we can identify certain types of relationships that are likely to frequently occur in MAS organizations. To give a few examples: a *control* relationship may identify an authority relationship of one role over another, in which a role can (partially) control the actions of another role; a *peer* relationships may express the fact that two roles have equal status; a *dependency* relation may express the fact that one role relies on some resources or knowledge from other roles for its accomplishment. In any case, these exemplar relationship types are neither mutually exclusive (e.g., a control relationship may also imply a dependency relationship and a dependency relationship may exist between peers), nor claim to define a complete set (other types of relationships can be identified).

In the manufacturing pipeline, and assuming that a collective organization has been chosen, all the stages are peers. This can be simply expressed as:

$$\forall i, j \quad \text{such that} \quad i \neq j, \textsc{Stage}[i] \xrightarrow{peer} \textsc{Stage}[j]$$

If the hierarchical organization is chosen instead, the coordinator is in charge of supporting all the stages for the protocol of speed negotiation. This structure expressing the dependencies of stages on the coordinator to negotiate the pipeline flux can be represent as:

$$\forall i, \textsc{Stage}[i] \xrightarrow{depends_on} \textsc{Coordinator}$$

In the conference management example, and assuming the choice of a multilevel hierarchy, the representation of the organizational structure is given below. The relations, respectively, express the fact that the PC Chair controls the papers assigned to the ReviewCatcher roles, that the ReviewCatcher role

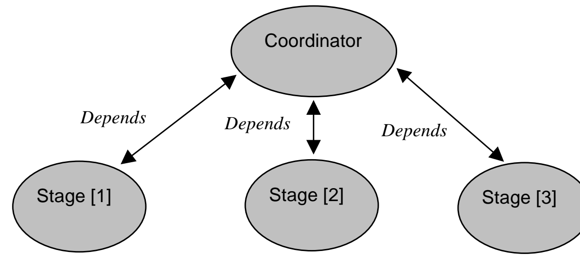


Fig. 14. The manufacturing pipeline: Representation of the hierarchical organizational structure.

controls REVIEWERS, and that REVIEWCATCHERS are peers with respect to the negotiation of papers.

$$\forall i, \text{PC CHAIR} \xrightarrow{control} \text{REVIEWCATCHER}[i]$$

$$\forall i,\ j, \text{REVIEWCATCHER}[i] \xrightarrow{control} \text{REVIEWER}[j]$$

$$\forall i,\ j, \text{REVIEWCATCHER}[i] \xrightarrow{peer} \text{REVIEWCATCHER}[j]$$

For all the above representations to be useful to developers, they must be properly detailed with a description of the semantics of the relations (i.e., of the protocol and activities involved in such relations, as detailed in the following subsection) and with any additional textual comments that could possibly enrich the notation and make it more comprehensible.

Since the formal representation of the relationship may not be sufficiently intuitive for all practitioners, a graphical representation may be used to complement it. Such a representation can simply reduce to representing roles in terms of "blocks," connected by annotated arrows, to represent relations and their types. Such a representation for the manufacturing example and the hierarchical organization is given in Figure 14. Coupling such a graphical representation with the one of the environmental model (representing in an abstract way the operational environment in which the MAS will be situated) can lead to a complete and intuitive graphical representation of all the entities involved in the MAS, of their interactions with each other, and with the operational environment.

Again, we emphasize that Gaia, per se, is not intended to commit to the adoption of a specific notation. With regard to organizational structure, besides the simple representational methods described above, other notations and graphical representation can be adopted to describe and represent roles and their interactions (e.g., AUML diagrams Bauer et al. [2001]).

4.2.4 *Completion of Role and Interaction Models.* Once the organizational structure is defined, the preliminary role and interaction models (as identified in the analysis phase) can be transformed into complete roles and interactions models (describing in detail the characteristics of all roles and stages involved in the MAS that will actually be produced). In fact, with the organizational structure identified, the designer knows which roles will have to interact with

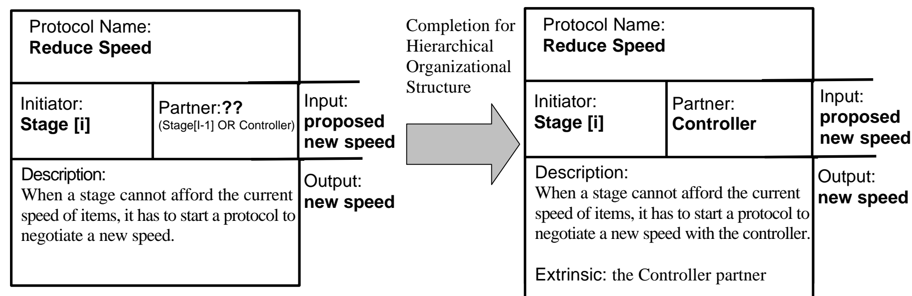


Fig. 15. The completion of the ReduceSpeed protocol.

which others (as derived from the organization topology) and which protocols will have to be executed (as derived from the control regime of the organization). Thus, the completion of the role and protocol model amounts to:

—defining all the activities in which a role will be involved, as well as its liveness and safety responsibilities;
—defining *organizational roles*—those whose presence was not identified in the analysis phase, and whose identification derives directly from the adoption of a given organizational structure;
—completing the definition of the protocols required by the application, by specifying which roles the protocol will involve;
—defining *organizational protocols*—those whose identification derives from the adopted organizational structure.

The complete definition of the roles and protocols can exploit the representation already introduced with respect to the preliminary roles and protocols (see Figures 9, 10, 11, 12).

With regard to the already identified preliminary roles and protocols, the difference is that the complete representation now includes all the characteristics of roles and protocols (i.e., for roles the full identification of activities and services, and for protocols the complete identification of the involved roles). As a simple example, Figure 15 shows how the preliminary ReduceSpeed protocol (represented in Figure 11) can be completed once a hierarchical organizational structure is chosen (so that the partner of any STAGE[*i*] initiating the protocol can be identified in the CONTROLLER leading the hierarchy).

With regard to roles and protocols whose definition directly derives from the adoption of a specific structure, their representation has to be defined from scratch (or by exploiting templates from catalogues of organizational patterns). As a simple example, Figure 16 shows the definition of a protocol Negotiate-SpeedReduction for the manufacturing pipeline application, whose introduction derives from having chosen a hierarchical organization: the CONTROLLER, once having been contacted by a STAGE[*i*] asking for a speed reduction, has to negotiate an acceptable new speed with all the other stages.

When completing the preliminary role and interaction models, it is important to clearly preserve the distinction between those characteristics that are

Protocol Name:
**EvaluateSpeedReduction**

| Initiator: **Controller** | Partner: **Stages[i], ∀ i** | Input: **proposed new speed** |
|---|---|---|
| Description: After a stage has requested a speed reduction, the Controller takes care of negotiating directly with all Stages a new acceptable speed for all of them<br><br>Extrinsic: the whole protocol is extrinsic, introduced from the fact of having elected a Controller to coordinate and mediate all speed reductions | | Output: **acceptable new speed** |

Fig. 16. The definition of the NegotiateSpeedReduction protocol.

"intrinsic" (i.e., independent of the use of the role/protocol in a specific organizational structure) and those characteristics that are "extrinsic" (i.e., derive from the adoption of a specific organizational structure). For instance, in the manufacturing pipeline example, the fact that the STAGE[*i*] roles initiate the ReduceSpeed protocol is intrinsic, whereas the fact that such a protocol involves the COORDINATOR role is extrinsic (since the COORDINATOR role is an organizational role, whose integration in the system derives from the choice of a hirarchical organizational structure). The above separation, that we have simply preserved as a comment to the description of the protocols (as in Figures 15 and 16), assumes a fundamental importance in the perspectives of reuse and design for change. If the definition of a role clearly specifies the logical separation between the two parts, and the implementation of the role somehow does the same, this enables the role be reused in different systems, independently of the specific organizational structure. This distinction also means that changes in the organizational structure can be made without forcing system developers to redesign and recode agents from scratch. Furthermore, if the implementation of the role, other than maintaining a clear separation between intrinsic and extrinsic characteristics also supports dynamic changes in the agent structure (as, for example, in frameworks supporting the run-time modification of the roles played by agents [Cabri et al. 2003]), the approach could promote the definition and implementation of systems capable of dynamically changing their internal organizational structure—by having agents readapt their extrinsic part to fit into a new organizational structure—in response to changed conditions. However, dealing with such dynamic reorganizations would need the multiagent system to monitor its execution and take decisions about its re-structuring and these are issues that are beyond the scope of this article and of the current version of Gaia.

Once completed, the roles and interaction models represent an operational description of the MAS organization that could be effectively exploited, possibly with the support of the environmental model (to get a better idea of the

characteristics of the MAS operational environment), for the detailed design of the MAS.

### 4.3 The Design Phase: Detailed Design

The detailed design phase is responsible for eventually identifying the agent model and the services model which, in turn, act as guidelines for the actual implementation of agents and their activities.

4.3.1 *Definition of the Agent Model.* In the Gaia context, an agent is an active software entity playing a set of agent roles. Thus, the definition of the agent model amounts to identifying which agent classes are to be defined to play specific roles and how many instances of each class have to be instantiated in the actual system.

Typically, as outlined in Section 4.2.1, there may be a one-to-one correspondence between roles and agent classes: a given role will be played by an instance of an agent of a class in charge of implementing that role. Such a correspondence naturally derives from the methodological process that was adopted in which, especially during the identification of the organizational structure, the concept of role has implicitly assumed ever greater concreteness. However, given that (i) the organizational efficiency is not affected, (ii) bounded rationality problems do not emerge, and (iii) this does not violate organizational rules, a designer can choose to package a number of closely related and strongly interacting roles in the same agent class for the purposes of convenience. There is obviously a trade-off between the *coherence* of an agent class (how easily its functionality can be understood), the efficiency considerations that come into play when designing agent classes, and the need to minimize mismatches with the real-world organization that the system intends to support.

With reference to the conference management example, it may be clear that it is desirable to have the same agent playing the role of ReviewCatcher and ReviewCollector. Such a choice affects neither the efficiency nor the bounded rationality of the agent, because the two roles are played at different times. In addition, such a choice does not conflict with any organizational rules. In contrast, such a choice may compact the whole design by reducing the number of agent classes and instances and it may also minimize the conceptual complexity (in that also in the real-world organization the two roles are typically played by the same person). With reference to the manufacturing pipeline, the choice of a one-to-one mapping of roles to agent classes is the most natural and it is not easy to think of a reasonable alternative. Thus, the system will be described by a Stage class and by a Coordinator class, whose instances will be in charge of implementing the roles with the same names.

The agent model of Gaia can be defined using a simple diagram (or table) specifying, for each agent class, which roles will map to it. In addition, the agent model can document the instances of a class that will appear in the MAS. This can be done by annotating agent classes with qualifiers from Fusion [Coleman et al. 1994]. An annotation $n$ means that there will be exactly $n$ agents of this class in the run-time system. An annotation $m..n$ means that there will be no less than $m$ and no more than $n$ instances of this class in a run-time system

($m < n$). An annotation $*$ means that there will be zero or more instances at run-time, and $+$ means that there will be one or more instances at run-time.

With reference to the conference management example, the notation:

$$\text{PC CHAIR}^1 \xrightarrow{play} \text{REVIEWCATCHER, REVIEWCOLLECTOR}$$

simply expresses that the agent class PC CHAIR will be defined to play both role REVIEWCATCHER and role REVIEWCOLLECTOR, and that a single instance of this class will be created in the MAS. In the case of the manufacturing pipeline, the notations:

$$\text{STAGE}^N \xrightarrow{play} \text{STAGE}$$

$$\text{COORDINATOR}^1 \xrightarrow{play} \text{COORDINATOR}$$

express the fact that the will be $N$ agents (one for each of the $N$ stages of the pipeline) of the class STAGE to play the STAGE role, and (if a hierarchical organizational structure has been chosen) a single agent of the class COORDINATOR to play the COORDINATOR role.

It is worth highlighting that the agent model of Gaia is of use in driving the static assignment of roles to agent classes, in which the implementation of the agent class is assumed to hardwire the code to implement one or more roles. Of course, if the implementation adopts a framework enabling agents to dynamically assume roles (as, for example, in the BRAIN framework [Cabri et al. 2003]), the concept of agent class of Gaia (and so the Gaia agent model) would no longer be of use.

As a final note, inheritance is given no part in Gaia's agent models, since our experience is that a MAS typically contains only a comparatively small number of roles and classes, making inheritance of little use in design. Of course, when it comes to actually *implementing* agents, inheritance will be used in the normal object-oriented fashion.

4.3.2 *The Services Model.* As its name suggests, the aim of the Gaia *services* model is to identify the services associated with each agent class or, equivalently, with each of the roles to be played by the agent classes. Thus, the services model applies both in the case of static assignment of roles to agent classes as well as in the case where agents can dynamically assume roles.

While in OO terms a service would roughly correspond to a method, agent services are not available for other agents in the same way that an object's methods are available for another object to invoke (as discussed in Section 2.2). Rather, due to the intrinsically active nature of an agent, a service is better thought of as a single, coherent block of activity in which an agent will be engaged. Such a service does not have to necessarily be triggered by external requests (i.e., the agent is autonomous in its activities), neither does an external request necessarily trigger a service (autonomy also implies there are internal decision capabilities).

For each service that may be performed by an agent, it is necessary to document its properties. Specifically, we must identify the *inputs*, *outputs*,

*preconditions* and *postconditions*. Inputs and outputs to services will be derived in an obvious way from both the protocols model (for services involving the elaboration of data and knowledge exchange between agents) and the environmental model (for services involving the evaluation and modification of environmental resources). Pre- and postconditions represent constraints on the execution and completion, respectively, of services. These are derived from the safety properties of a role, as well as from the organizational rules, and may involve constraints on the availability and the specific values assumed by either environmental resources or data and knowledge from other agents.

The services that compose an agent are derived from the list of protocols, activities, responsibilities and liveness properties of the roles it implements. At one extreme, there will be at least one service for each parallel activity of execution that the agent has to execute. However, even for sequential activities of execution, there may be a need to introduce more services to represent different phases of the agent execution. Let us return to the conference management example and, specifically, to the REVIEWER role. There are three activities and protocols associated with this role: ReceivePaper, ReviewPaper, and SendReview. As these protocols and activities are to be performed sequentially during the agent execution, it is possible to think of defining the agent in terms of a single service, to be activated upon request (i.e., upon inititation by a REVIEWCATCHER of the ReceivePaper protocol), and eventually leading to the execution of the SendReview protocol. The input of the service is a request for reviewing a paper. The output of the service is the completed review form. The preconditions on the execution of the service could be that the same agent has not already executed that service *maximum_number* times (i.e., is not already reviewing the maximum allowed number of papers) and that the paper is available for reading in the environment. The postcondition is that the review form has been correctly completed. However, since the outcome of the ReceivePaper protocol may be different (e.g., a reviewer may decline to review a paper), the design would be better to split the larger service into two smaller ones. This may separate the service associated with the execution of the ReceivePaper protocol from the one associated with the paper review, the latter being executed after a paper has been accepted for review.

The Gaia services model does not prescribe an implementation for the services it documents. Rather the developer is free to realise the services in any implementation framework deemed appropriate. For example, it may be decided to implement services directly as methods in an object-oriented language. Alternatively, a service may be decomposed into a number of methods.

### 4.4 Outcome of the Design Phase

After the successful completion of the Gaia design process, developers are provided with a welldefined set of agent classes to implement and instantiate, according to the defined agent and services model. As already stated, Gaia does not deal with implementation issues and considers the output of the design phase as a specification that can be picked up by using a traditional method or that could be implemented using an appropriate agent-programming

framework. These considerations rely on the assumption that specific technological constraints (e.g., those that may occur with regard to modeling and accessing to the environment) are identified earlier in the process.

As an additional note, although the above methodology is explicitly conceived for open agent systems, its outcomes can also be exploited by:

— third-party developers of agents, whenever they intend to have their agents execute in other organizations, as can be the case of agents that are used in agent-based marketplaces;
— agent users, whenever they wish to exploit their personal assistant agents in the context of specific workgroup applications.

In both cases, assuming that the developers of the existing MAS have made all the Gaia models publicly available, an agent developer will be able to check whether their agent can enter the system and, if so, what rules and procedures it needs to conform to.

## 5. RELATED WORK

Research in the area of agent-oriented software engineering has expanded significantly in the past few years. Several groups have started addressing the problem of modeling agent systems with appropriate abstractions and defining methodologies for MASs development.

### 5.1 Modeling Abstractions

Several researchers in this area have defined modeling techniques specifically tuned to the abstractions promoted by agents and MASs (as discussed in Section 2.1), without attempting to define complete methodologies for the development of MASs.

As stated in Section 1, we believe conventional analysis and design methodologies, such as object-oriented ones [Booch 1994], are ill suited to MASs because of the fundamental mismatch between the abstractions they provide [Wooldridge et al. 2000]. Consequently, we believe that those efforts in the area of agent-based computing that attempt to describe agents and MASs by simply *applying* well-assessed object-oriented modeling techniques (e.g., Kendall [2001]) will inevitably fall short. For instance, although it may be possible to represent concepts such as organizational rules and complex interaction protocols in terms of the abstractions of, say, object-oriented computing, this is a convoluted process and one that lowers the abstraction level. Nevertheless, we are aware that the adoption of well-known notations and modeling techniques might facilitate the rapid diffusion of agent-based computing as a software engineering paradigm.

For these reasons, we feel that a more promising approach is the attempt to *extend* well-assessed abstractions and modeling techniques. For instance, the AUML effort [Odell et al. 2001; Bauer et al. 2001] starts from UML [Rumbaugh et al. 1998] and extends it with additional abstractions and notations specifically tuned to the particular problems of agent-based applications. However, to date, AUML has only achieved good results in modeling complex interaction protocols between components. It still lacks appropriate models for complex

organizational structures and the associated organizational laws [Bauer et al. 2001]. Nevertheless, AUML is gaining acceptance in the area and a large amount of work is being devoted to finding suitable ways of extending it further to make it more suitable. Therefore, as it matures, AUML is likely to become a useful companion to Gaia.

Besides these object-oriented notations, a number of agent-specific modeling abstractions and techniques have been proposed in recent years (see Iglesias et al. [1999] for a survey and Wooldridge [1997] for a discussion). Several of these attempt to exploit the idea of a MAS as a computational organization. In most cases, these proposals define an organization simply as a collection of roles (i.e., a role model). This is what happens, for example, in the AALAADIN system [Ferber and Gutknecht 1998], which models MASs in terms of organizations where "the group structure," characterizing organizations, is simply the collection of roles that compose them. Analogously, in the ToolKit approach [Demazeau and Rocha Costa 1996], an organization is defined simply by the set of roles that compose it and by the interaction protocols that have to occur between roles. Neither of these approaches incorporate the notions of organizational rules or organizational structures and, for the reasons we have already outlined in Section 3, will be limited in the range of agent systems they can deal with.

A limited amount of work explicitly addresses the problem of defining organizational rules to orchestrate the behavior of a multiagent organization. However, one application area in which there is a particularly strong need for organizational rules is that of computational markets. For systems in which heterogeneous agents meet to perform commercial transactions, it is important to identify and enact rules to guarantee that the overall activity of the market can correctly progress, despite possibly opportunistic behavior by participants [Esteva et al. 2001]. Such considerations were behind the implementation of the Fishmarket framework for agent-mediated auctions [Noriega 1997]. Here, there is a need to compel agents to act in accordance with the social conventions that rule the organization of an auction. This is then implemented in terms of controller agents associated with each of the agents participating in an auction. Other agent-based approaches for Web-based workgroup applications start from similar considerations [Ciancarini et al. 2000; Cabri et al. 2002], and conceive interactions between agents as occurring via "active environments" whose behavior can be modeled so as to implement specific policies for governing agent interactions. In other words, active environments are exploited as the repository of organizational rules. Our work differs from these endeavors in that we focus on the particular organizational abstractions that are appropriate for the software engineering process, rather than on the specific technologies to implement them.

### 5.2 Agent-Oriented Methodologies

Several complete methodologies for the analysis and design of MASs have been proposed so far (see Shehory and Sturm [2001] for a survey). However, very few of them explicitly focus on organizational abstractions.

The MASE Methodology [Wood et al. 2001] provides guidelines for developing MASs based on a multistep process. In analysis, the requirements are used to define use-cases and application goals and subgoals, and eventually to identify the roles to be played by the agents and their interactions. In design, agent classes and agent interaction protocols are derived from the outcome of the analysis phase, leading to a complete architecture of the system. However, the MASE process fails to identify any organizational abstraction other than the role model: there are no abstractions exploited in the analysis phase that can be assimilated to organizational rules, and the definition of the organizational structure is derived as an implicit outcome of the analysis phase. As explicitly acknowledged by the MASE designers, this makes MASE suitable only for closed agent systems.

The MESSAGE methodology [Caire et al. 2002] exploits organizational abstractions that can be mapped into the abstractions identified by Gaia. In particular, MESSAGE defines an organization in terms of a *structure*, determining the roles to be played by the agents and their topological relations (i.e., the interactions occurring among them). This corresponds to the concept of organizational structure promoted by Gaia. In addition, in MESSAGE, an organization is also characterized by a *control entity* and by a *workflow structure*. These two concepts, when taken together, map into Gaia's concept of organizational rules and they determine the laws that agents' actions and interactions have to conform to during execution. Still, MESSAGE does not address the problem of identifying and explicitly modeling the organizational rules (they are implicitly woven into the organizational structure) and it does not recognize the need to explicitly design the organizational structure (that is assumed as an implicit outcome of the analysis phase). As an additional note, MESSAGE is tightly bound to UML (and to AUML), while Gaia is more general, and provides clear guidelines without committing to any specific modeling techniques.

The Gaia methodology described in this article is an extension of the version described in Wooldridge et al. [2000]. The first version of Gaia, as is the case with the current one, provided a clean separation between the analysis and design phases. However, as already noted in this article, it suffered from limitations—similar to the ones of MESSAGE and MASE—caused by the incompleteness of its set of abstractions. The objective of the analysis phase in the first version of Gaia was to define a fully elaborated role model, derived from the system specification, together with an accurate description of the protocols in which the roles will be involved. This implicitly assumed that the overall organizational structure was known a priori (which is not always the case). In addition, by focusing exclusively on the role model, the analysis phase in the first version of Gaia failed to identify both the concept of global organizational rules (thus making it unsuitable for modeling open systems and for controlling the behavior of self-interested agents) and the modeling of the environment (which is indeed important, as extensively discussed in this article). The new version of Gaia overcomes these limitations.

A further extension to the first version of Gaia is also proposed in Juan et al. [2002] and is aimed at addressing the same limitations (i.e., representation of the environment and organizational rules). However, that

proposal is still rather preliminary and lacks a coherent integration in the Gaia process.

The TROPOS methodology, first proposed in Bresciani et al. [2001] and refined in Kolp et al. [2002], shares with Gaia both the adoption of the organizational metaphor and an emphasis on the explicitly study and identification of the organizational structure. Like Gaia, TROPOS recognizes that the organizational structure is a primary dimension for the development of agent systems and that an appropriate choice of it is needed to meet both functional and non-functional requirements. However, TROPOS also differs from Gaia in several aspects. On the one hand, the TROPOS methodology defines uniform and coherent guidelines for the activities of both early and late requirements engineering, while Gaia's analysis mostly deals with late requirements engineering. However, as already stated, we do not exclude the possibility of integrating in Gaia early requirements analysis, possibly adapting (as in TROPOS) methods and techniques from goal-oriented analysis [Mylopoulos et al. 1999; Castro et al. 2002]. On the other hand, TROPOS's analysis does not explicitly identify the concept of organizational rules. Although some explicit structural dependencies between roles are required to be identified in the analysis phase, no TROPOS model is able to capture global laws that apply to multiple organizational roles or to the organization as a whole. As already discussed in the paper, this may undermine the effectiveness of the analysis and increase the complexity of the subsequent design phase.

## 6. CONCLUSIONS AND FUTURE WORK

This article has argued that agent-oriented computing is an appropriate software engineering paradigm for the analysis, design, and development of many contemporary software systems. In particular, we focused on the key issues related to the identification of appropriate abstractions for agent-based software engineering and to the definition of a suitable methodology for the analysis and design of complex applications in terms of multiagent systems. In so doing, the main contributions of this article are:

— a clarification of the relationship between the agent-oriented approach to software development and that of more traditional (e.g., object- and component-based) approaches;
— the identification of a suitable set of core abstractions, inspired by an organizational metaphor, to be used during the analysis and design of multiagent systems; and
— the development of a clear methodology, centered around organizational abstractions, for the analysis and design of open multiagent systems.

While this is an important step towards the widespread acceptance of agent-based computing as a software engineering paradigm much work remains to be done. In particular:

— robust tools, programming models and development environments that embody the appropriate set of agent abstractions need to be made widely available and easy to use by software engineering practitioners;

—catalogues of organizational patterns (to guide the definition of the organizational structure) and libraries of reusable design components (to guide the detailed design of agents) need to be developed and disseminated in a widely accessible form;

—while the presented Gaia methodology suggests a rather sequential approach to software development, proceeding linearly from analysis to design and implementation, software development have often to deal with unstable requirements, wrong assumptions, and missing information. This requires the designer to step back from the current activities and, perhaps, to rethink some of the decisions. As it stands, however, there are no explicit structures or processes in Gaia for doing this.

## REFERENCES


ABELSON, H., ALLEN, D., COORE, D., HANSON, C., HOMSY, G., KNIGHT, T., NAPAL, R., RAUCH, E., SUSSMANN, G., AND WEISS, R. 2000. Amorphous computing. *Commun. ACM 43*, 5 (May), 43–50.

BABAOGLU, O., MELING, H., AND MONTRESOR, A. 2002. Anthill: A framework for the development of agent-based peer-to-peer systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems* (Vienna, Austria). IEEE Computer Society Press, Los Alamitos, Calif., pp. 15–22.

BASS, L., CLEMENTS, P., AND KAZMAN, R. 2003. *Software Architectures in Practice* (2nd Ed.). Addison-Wesley, Reading, Mass.

BAUER, B., MULLER, J. P., AND ODELL, J. 2001. Agent UML: A formalism for specifying multiagent software systems. *Int. J. Softw. Eng. Knowl. Eng. 11*, 3 (Apr.), 207–230.

BERNERS-LEE, T., HENDLER, J., AND LASSILA, O. 2001. The semantic web. *Sci. Amer.* May.

BONABEAU, E., DORIGO, M., AND THERAULAZ, G. 1999. *Swarm Intelligence. From Natural to Artificial Systems*. Oxford University Press, Oxford, U.K.

BOOCH, G. 1994. *Object-Oriented Analysis and Design* (2nd ed.). Addison-Wesley, Reading, Mass.

BRESCIANI, P., PERINI, A., GIORGINI, P., GIUNCHIGLIA, F., AND MYLOPOULOS, J. 2001. A knowledge level software engineering methodology for agent oriented programming. In *Proceedings of the 5th International Conference on Autonomous Agents* (Montreal, Ont., Canada, June). ACM, New York, pp. 648–655.

BUSSMANN, S. 1998. Agent-oriented programming of manifacturing control tasks. In *Proceedings of the 3rd International Conference on Multi-Agent Systems* (Paris, France, June). IEEE Computer Society Press, Los Alamitos, Calif., pp. 57–63.

CABRI, G., FERRARI, L., AND LEONARDI, L. 2003. Enabling mobile agents to dynamically assume roles. In *Proceedings of the 2003 ACM Symposium on Applied Computing* (Melbourne, Fla., Mar.). ACM, New York, pp. 56–60.

CABRI, G., LEONARDI, L., AND ZAMBONELLI, F. 2000. Mobile-agent coordination models for internet applications. *IEEE Comput. 33*, 2 (Feb.), 52–58.

CABRI, G., LEONARDI, L., AND ZAMBONELLI, F. 2002. Engineering mobile agent applications via context-dependent coordination. *IEEE Trans. Softw. Eng. 28*, 11 (Nov.), 1034–1051.

CAIRE, G., COULIER, W., GARIJO, F., GOMEZ, J., PAVON, J., LEAL, F., CHAINHO, P., KEARNEY, P., STARK, J., EVANS, R., AND MASSONET, P. 2002. Agent-oriented analysis using message/uml. In *Proceedings of the 2nd International Workshop on Agent-Oriented Software Engineering*. Lecture Notes in Computer Science, vol. 2222. Springer Verlag, New York, pp. 119–135.

CASTRO, J., KOLP, M., AND MYLOPOULOS, J. 2002. Towards requirements-driven information systems engineering: The tropos project. *Inf. Syst. 27*, 6 (June), 365–389.

CIANCARINI, P., OMICINI, A., AND ZAMBONELLI, F. 2000. Multiagent systems engineering: The coordination viewpoint. In *Intelligents Agents VI: Agent Theories, Architectures, and Languages*. Lecture Notes in Computer Science, vol. 1767. Springer-Verlag, New York, pp. 250–259.

COLEMAN, D., ARNOLD, P., BODOFF, S., DOLLIN, D., GILCHRIST, H., HAYES, F., AND JEREMAS, P. 1994. *Object-Oriented Development: The FUSION Method*. Prentice-Hall International, Hemel Hampstead U.K.

COLLINOT, A., DROGOUL, A., AND BENHAMOU, P. 1996. Agent-oriented design of a soccer robot team. In *Proceedings of the 2nd International Conference on Multi-Agent Systems* (Kyoto, Japan). IEEE Computer Society Press, Los Alamitos, Calif.

DEMAZEAU, Y., AND ROCHA COSTA, A. C. 1996. Populations and organizations in open multi-agent systems. In *Proceedings of the 1st National Symposium on Parallel and Distributed AI*.

ESTEVA, M., RODRIGUEZ-AGUILAR, J. A., SIERRA, C., GARCIA, P., AND ARCOS, J. L. 2001. On the formal specifications of agent institutions. In *Agent-Mediated Electronic Commerce*. Lecture Notes in Computer Science, vol. 1991. Springer-Verlag, New York, pp. 126–147.

ESTRIN, D., CULLER, D., PISTER, K., AND SUKJATME, G. 2002. Connecting the physical world with pervasive networks. *IEEE Perv. Comput. 1*, 1, 59–69.

FERBER, J. AND GUTKNECHT, O. 1998. A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of the 3rd International Conference on Multi-Agent Systems* (Paris, France). IEEE Computer Society Press, Los Alamitos, Calif., pp. 128–135.

FOSTER, I. AND KESSELMAN, C. (EDS.). 1999. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, San Francisco, Calif.

FOX, M. S. 1981. An organizational view of distributed systems. *IEEE Trans. Syst. Man, Cybernet. 11*, 1 (Jan.), 70–80.

HANDY, C. 1976. *Understanding Organizations*. Penguin Books, London, UK.

HATTORI, F., OHGURO, T., YOKOO, M., MATSUBARA, S., AND YOSHIDA, S. 1999. Socialware: Multiagent systems for supporting network communities. *Communi. ACM 42*, 3 (Mar.), 55–61.

IGLESIAS, C., GARIJO, M., AND GONZALES, J. 1999. A survey of agent-oriented methodologies. In *Intelligents Agents IV: Agent Theories, Architectures, and Languages*. Lacture Notes in Artificial Intelligence, vol. 1555. Springer-Verlag, New York, pp. 317–330.

JENNINGS, N. R. 2001. An agent-based approach for building complex software systems. *Commun. ACM, 44*, 4 (Apr.), 35–41.

JUAN, T., PIERCE, A., AND STERLING, L. 2002. Roadmap: Extending the gaia methodology for complex open systems. In *Proceedings of the 1st ACM Joint Conference on Autonomous Agents and Multi-Agent Systems* (Bologna, Italy, July). ACM, New York, pp. 3–10.

KENDALL, E. A. 2001. Agent software engineering with role modelling. In *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering*. Lecture Notes in Computer Science, vol. 1957. Springer-Verlag, New York, pp. 163–170.

KEPHART, J. O. AND CHESS, D. M. 2003. The vision of autonomic computing. *IEEE Comput. 36*, 1, (Jan.), 41–50.

KOLP, M., GIORGINI, P., AND MYLOPOULOS, J. 2002. A goal-based organizational perspective on multiagent architectures. In *Intelligent Agents VIII: Agent Theories, Architectures, and Languages*. Lecture Notes in Artificial Intelligence, vol. 2333. Springer-Verlag, New York, pp. 128–140.

LEER, G. AND HEFFERMAN, D. 2002. Expanding automotive electronic systems. *IEEE Comput. 35*, 1 (Jan.), 88–93.

LIND, J. 2001. *Iterative Software Engineering for Multiagent Systems: The MASSIVE Method*. Lecture Notes in Computer Science, vol. 1994. Springer-Verlag, Berlin, Germany.

MAES, P. 1994. Agents that reduce work and information overload. *Commun. ACM 37*, 7 (July), 30–40.

MAMEI, M., ZAMBONELLI, F., AND LEONARDI, L. 2003. Distributed motion coordination in co-fields. In *Proceedings of the 6th Symposium on Autonomous Decentralized Systems* (Pisa, Italy, Apr.). IEEE Computer Society Press, Los Alamitos, Calif., pp. 63–70.

MANNA, Z. AND PNUELI, A. 1995. *Temporal Verification of Reactive Systems—Safety*. Springer-Verlag, Berlin, Germany.

MATARIC, M. 1992. Integration of representation into goal-driven behavior-based robots. *IEEE Trans. Rob. Automat. 8*, 3 (Dec.), 59–69.

MINTZBERG, H. 1979. *The Structuring of Organizations: A Synthesis of the Research*. Prentice-Hall, Englewood Cliffs, N.J.

MOSES, Y. AND TENNENHOLTZ, M. 1995. Artificial social systems. *Comput. Artif. Intel. 14*, 3, 533–562.

MYLOPOULOS, J., CHUNG, L., AND YU, E. 1999. From object-oriented to goal-oriented requirements analysis. *Communi. ACM 42*, 1 (Jan.), 31–37.

NORIEGA, P. 1997. *Agent-mediated Auctions: The Fishmarket Metaphor*. Ph.D Thesis, Universitat Autonoma de Barcelona, Barcelona, Spain.

ODELL, J., VAN DYKE PARUNAK, H., AND BOCK, C. 2001. Representing agent interaction protocols in UML. In *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering*. Lecture Notes in Computer Science, vol. 1957. Springer-Verlag, New York, pp. 121–140.

PARUNAK, V. 1997. Go to the ant: Engineering principles from natural agent systems. *Ann. Oper. Res. 75*, 69–101.

RICCI, A., OMICINI, A., AND DENTI, E. 2002. Agent coordination infrastructures for virtual enterprises and workflow. *Int. J. Coop. Inf. Syst. 11*, 3 (Sept.), 335–380.

RIPEANI, M., IAMNITCHI, A., AND FOSTER, I. 2002. Mapping the gnutella network. *IEEE Internet Comput. 6*, 1 (Jan.), 50–57.

RUMBAUGH, J., JACOBSON, I., AND BOOCH, G. 1998. *The Unified Modeling Language Reference Manual*. Addison-Wesley, Reading, Mass.

SCHWABE, D., GUIMARAER, R. M., AND ROSSI, G. 2002. Cohesive design of personalized web applications. *IEEE Internet Comput. 6*, 2 (Apr.), 34–43.

SHAW, M., DELINE, R., KLEIN, D., ROSS, T., YOUNG, D., AND ZELESNIK, G. 1995. Abstractions for software architecture and tools to support them. *IEEE Trans. Softw. Eng. 21*, 4 (Apr.), 314–335.

SHAW, M. AND GARLAN, D. 1996. *Software Architectures: Perspectives on an Emerging Discipline*. Prentice-Hall, Englewood Cliffs, N.J.

SHEHORY, O. AND STURM, A. 2001. Evaluation of modeling techniques for agent-based systems. In *Proceedings of the 5th International Conference on Autonomous Agents* (Montreal, Ont., Canada, June). ACM, New York, pp. 624–631.

SIMON, H. A. 1957. *Models of Man*. Wiley, New York.

SMITH, R. G. 1980. The contract net protocol: High level communication and control in a distributed problem solver. *IEEE Trans. Comput. 29*, 12 (Dec.), 1104–1113.

TAHARA, Y., OHSUGA, A., AND HONIDEN, S. 1999. Agent system development based on agent patterns. In *Proceedings of the 21st International Conference on Software Engineering* (Los Angeles, Calif., May). ACM, New York, pp. 356–367.

TENNENHOUSE, D. 2000. Embedding the Internet: Proactive computing. *Commun. ACM 43*, 5 (May), 36–42.

WOOD, M., DELOACH, S. A., AND SPARKMAN, C. 2001. Multiagent system engineering. *Int. J. Softw. Eng. Knowl. Eng. 11*, 3 (Apr.), 231–258.

WOOLDRIDGE, M. 1997. Agent-based software engineering. *IEE Proc. Softw. Eng. 144*, 1 (Feb.), 26–37.

WOOLDRIDGE, M. 2000. *Reasoning about Rational Agents*. MIT Press, Cambridge, Mass.

WOOLDRIDGE, M. 2002. *An Introduction to Multiagent Systems*. Wiley, New York.

WOOLDRIDGE, M. AND JENNINGS, N. R. 1995. Intelligent agents: Theory and practice. *Knowl. Eng. Rev. 10*, 2, 115–152.

WOOLDRIDGE, M., JENNINGS, N. R., AND KINNY, D. 2000. The Gaia methodology for agent-oriented analysis and design. *J. Autonom. Agents Multi-Agent Syst. 3*, 3, 285–312.

ZAMBONELLI, F., JENNINGS, N. R., OMICINI, A., AND WOOLDRIDGE, M. 2001a. Agent-oriented software engineering for internet applications. In *Coordination of Internet Agents: Models, Technologies, and Applications*. Springer-Verlag, Berlin, Germany, pp. 326–346.

ZAMBONELLI, F., JENNINGS, N. R., AND WOOLDRIDGE, M. 2001b. Organizational rules as an abstraction for the analysis and design of multi-agent systems. *Int. J. Softw. Knowl. Eng. 11*, 3 (Apr.), 303–328.

ZAMBONELLI, F. AND PARUNAK, H. V. D. 2003. Signs of a revolution in computer science and software engineering. In *Proceedings of the 3rd International Workshop on Engineering Societies in the Agents World*. Lecture Notes in Computer Science, vol. 2577. Springer-Verlag, New York, pp. 13–28.

Received October 2002; revised September 2003; accepted October 2003