International Journal of Software Engineering and Knowledge Engineering
Vol. 0, No. 0 (1994) 000–000

Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems

FRANCO ZAMBONELLI
Dipartimento di Scienze dell'Ingegneria, Università di Modena e Reggio Emilia
Via Vignolese 905 – 41100 Modena, Italy
franco.zambonelli@unimo.it

and

NICHOLAS R. JENNINGS
Dept of Electronics and Computer Science, University of Southampton
Southampton SO17 1BJ, United Kingdom
nrj@ecs.soton.ac.uk

and

MICHAEL WOOLDRIDGE
Dept of Computer Science, University of Liverpool
Liverpool L69 7ZF, United Kingdom
M.J.Wooldridge@csc.liv.ac.uk

Received (received date)
Revised (revised date)
Accepted (accepted date)

Multi-agent systems can very naturally be viewed as computational organisations. For this reason, we believe organisational abstractions offer a promising set of metaphors and models that can be exploited in the analysis and design of such systems. To this end, the concept of role models is increasingly being used to specify and design multi-agent systems. However, this is not the full picture. In this paper we introduce three additional organisational concepts — *organisational rules*, *organisational structures*, and *organisational patterns* — and discuss why we believe they are necessary for the complete specification of computational organisations. In particular, we focus on the concept of organisational rules and introduce a formalism, based on temporal logic, to specify them. This formalism is then used to drive the definition of the organisational structure and the identification of the organisational patterns. Finally, the paper sketches some guidelines for a methodology for agent-oriented systems based on our expanded set of organisational abstractions.

Keywords: Organisational abstractions; multi-agent systems; agent-oriented software engineering; computational organisations

1. Introduction

Autonomous agents and multi-agent systems (MASs) are rapidly emerging as a powerful paradigm for designing and developing complex software systems [17, 36]. Therefore, as is the case with any new software engineering paradigm, the successful and widespread deployment of MASs requires not only new models and technologies, but also new *methodologies* to support developers engineer such systems in a robust, reliable, and repeatable fashion. In the last few years, there have been several attempts to develop such methodologies. However, most of this work is either tuned to specific systems and agent architectures [10, 5] — thus it lacks generality — or it is defined as an extension of existing object-oriented methodologies [15] — thus it exploits abstractions that are unsuitable for modelling agent-based systems.

Against this background, there are comparatively few proposals that attempt to define complete and general methodologies, *specifically tailored to the analysis and design of MASs.* One such methodology is Gaia [35]. Gaia views the process of analysing and designing multi-agent systems as one of constructing computational organisations. Thus, multi-agent systems are viewed as being composed of a number of autonomous interacting entities (an *organised society* of individuals) in which each agent plays one (or more) specific *roles.* In particular, Gaia, like a number of other agent-oriented methodologies [10, 9, 18], suggests defining the structure of a MAS in terms of a *role model.* This model identifies the roles that agents have to play within the MAS and the interaction protocols in which the different roles are involved.

The adoption of a role model as the main organisational abstraction means that the above mentioned methodologies are best suited to MASs in which the agents are inherently cooperative towards one another and in which the system is closed. However, to deal with the more general case in which systems contain self-interested agents operating in an open environment, we believe that additional organisational abstractions are necessary [37]. In particular, we believe that *organisational rules*, *organisational structures*, and *organisational patterns* must also play a primary role in the analysis and design of such MASs. Organisational rules express general, global (supra-role) requirements for the proper instantiation and execution of a MAS. An organisational structure defines the specific class (among the many possibilities) of organisation and control regime to which the agents and roles have to conform in order for the whole MAS to work efficiently and according to its specified requirements. Organisational patterns express pre-defined and ubiquitous organisational structures that can be re-used from system to system (in a manner similar to the way catalogues of patterns are widely exploited in the design of object-oriented systems [12]).

Of these three new organisational abstractions, this paper focuses predominantly on the notion of organisational rules. The correct identification of such rules in the analysis phase is of fundamental importance. Indeed, organisational rules strongly influence the design phase in choosing the organisational structures and in deciding to exploit specific organisational patterns. Here we explore the concept of organisational rules in detail and introduce a formalism, based on temporal logic [23, 24], for their specification. Temporal logic was chosen because it is well suited to expressing requirements related to the structure of a multi-agent system and to its evolution over time, and it can be effectively exploited to support designers in the definition of the organisational structure and in the identification of the appropriate organisational patterns. Speaking more generally, the appropriate formalisation of the organisational abstractions also leads to more clean, manageable, and re-usable MAS designs.

The paper is organised as follows. Section 2 introduces the basic concepts underlying agents and multi-agent systems, and introduces two application examples that will be taken as case studies through the paper to exemplify the concepts expressed. Section 3 introduces the additional organisational abstractions that are needed for a methodology to apply to open systems, and motivates their adoption. Section 4 details the concept of organisational rules and shows how they can be effectively expressed in temporal logic. Section 5 briefly sketches how our organisational abstractions can be exploited in the context of a new methodology for the analysis and design of MASs. Section 5 discusses related work in this area and, finally, section 6 concludes by outlining the open issues and the future research directions.

2. Multi-Agent Systems and Organisations

Agents are software entities that exhibit *autonomous* and *proactive* goal-directed behaviour — their activities are not subject to a global flow of control and they can take the initiative where appropriate — and that are *reactive* to changes in the environment in which they are situated [34, 21]. These characteristics make agents useful as stand-alone entities that are delegated to accomplish a given task on behalf of a user (e.g., personal digital assistants, e-mail filterers, or simple robots). However, in the majority of cases, agents exist in the context of *multi-agent software systems*, whose global behaviour derives from the interaction among the constituent agents [14]. In these cases, agents also exhibit *social* behaviour; they interact with one another: either cooperating to achieve a common objective or helping each other to achieve their individual objectives.

Here, we distinguish between two main classes of multiple agent system: *(i) distributed problem solving systems* in which the component agents are explicitly designed to cooperatively achieve a given goal, and *(ii) open systems* in which agents, not necessarily co-designed to share a common goal, can dynamically leave and enter the system. In the former case, all agents are known a priori, and all agents are supposed to be benevolent to each other and, therefore, they can trust one another during interactions. In the latter case, the dynamic arrival of unknown agents needs to be taken into account, as well as the possibility of self-interested behaviour in the course of the interactions.

2.1. *The Organisational Metaphor*

The design of parallel and distributed applications, as well as of distributed object systems, usually relies on an architecture that derives from the decomposition of the functionalities and data required by the system to achieve its goal, and on the definition of their inter-dependencies [2]. In MASs, however, the autonomous and proactive behaviour of the constituent agents suggests that applications can be better designed by drawing inspiration from the behaviour and structure of human organisations. Thus, each agent is assigned a specific role in the system; that is, a well-defined task or responsibility in the context of the overall system, that the agent has to accomplish in an autonomous fashion, without any centralised control. With this view, interactions are no longer merely an expression of inter-dependencies, rather they are seen as a means by which an agent can accomplish its role in the organisation. Therefore, interactions are clearly identified and localised in the definition of the role itself, and they help characterise the position of the agent in the organisation.

An organisational perspective can also make the design of the system less complex and easier to manage than more traditional metaphors for concurrent systems. Firstly, each agent becomes a separate *locus* of control, in charge of accomplishing its role and being fully responsible for it. Secondly, since agents typically embed most of the functionality they need to accomplish their role, inter-dependencies between the system components are reduced. When taken together, these points ease the design process because they lead to a cleaner separation between the component-level (i.e., intra-agent) and system-level (i.e., inter-agent) design dimensions.

A final advantage relates to the fact that, in many cases, MASs are intended to support and/or control some real-world organisation. For example, MASs can be adopted to support the workflow management in a team or to help control the activities of an Internet auction. In such cases, an organisation-based MAS design reduces the conceptual distance between the software system and the real-world system it has to support. This, in turn, simplifies the development of the system.

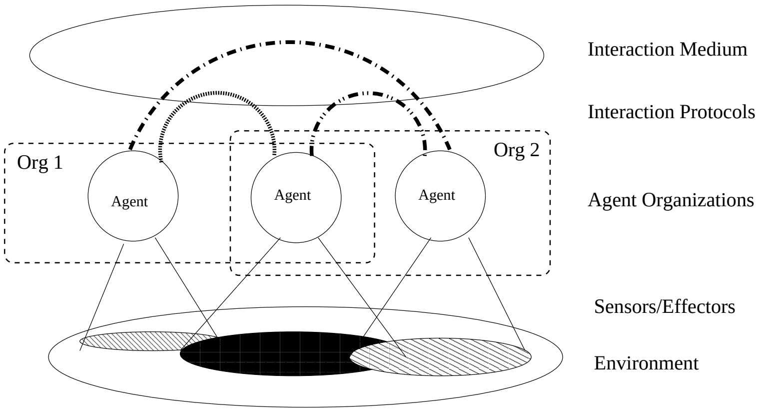

Figure 1: Characterisation of a Multi-Agent System

2.2. *An Organisational Characterisation of Multi-Agent Systems*

The organisational perspective leads to a general characterisation of a MAS as depicted in figure 1 [9, 16]. Although some simpler systems can be viewed as a single organisation, as soon as the complexity increases, *modularity* and *encapsulation* principles suggest splitting the system into different sub-organisations. Thus, in most cases, a complex multi-agent system can be viewed as several *interacting organisations*. Naturally a given agent can be part of multiple organisations.

In each organisation, an agent can play one or more *roles*. The role is what the agent is expected to do in the organisation: both in cooperation with the other agents and in respect of the organisation itself. Often, the role of an agent is simply defined in terms of the specific task that the agent has to accomplish in the context of the overall organisation. However, in our work, the notion of a role is much more precise, in that it gives an agent a well-defined position in the organisation, with an associated set of expected behaviours.

To accomplish their role in the organisation, agents typically need to *interact* with each other in order to *exchange knowledge* and *coordinate* their activities. Therefore, the concept of inter-agent interaction is strictly related to the agent's role. It is the role that requires a given form of interaction. Even more precisely, an agent, by the very fact that it plays a given role and has a well-defined position in the organisation, is committed to certain interaction protocols with the other agents in the organisation. Of course, the need to interact according to specific protocols requires the presence of an interaction medium between agents. This can either be a traditional network infrastructure, typically enforcing a message-passing interaction model, or another infrastructure possibly enforcing a different interaction model (e.g., a shared dataspace enforcing an indirect, data-oriented interaction model [13]).

Generally speaking, a MAS is immersed in a given *environment* with which the agents may need to interact in order to accomplish their role. This interaction occurs via *sensors* and *effectors*, i.e., mechanisms that enable agents to sense and effect a selected portion of the environment. That portion of the environment that an agent can sense and effect is determined by the agent's specific role, as well as on its current status.

2.3. *Exemplar Multi-Agent Systems*

To help illustrate our arguments about the need for organisational abstractions, we will consider two sample problems that will act as running examples throughout this paper.

Manufacturing Pipeline: As an example of the class of distributed problem solvers, we consider a system for the control of a manufacturing process. Specifically, let us consider the process of assembling, painting and packing metal hardware. Typically, such a control system can be delegated to a multiplicity of agent organisations, each devoted to the control of a well-defined portion of the overall manufacturing process (e.g., the assembling section or the painting section). Within each section, agents can then be associated with the control of a specific tool in the control system or to the control of a specific condition that must be assured to guarantee the correctness of the process.

In this context, we specifically consider a manufacturing pipeline in which items are transformed or augmented (e.g., a pipeline in which metal items are painted). Here, different agents can be devoted to the control of different stages of the pipeline (e.g., an agent is devoted to controlling the paint spraying, another is devoted to controlling the heat treatment of the paint, another of controlling the cooling process). Agents interact both indirectly through the environment and directly, through various forms of interaction protocol. In such an organisation, the role of each agent is that of "stage of the pipeline", 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. In addition, the agents need to interact 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 an open system we consider an agent-based system for supporting the management of an international conference. Setting up and running a conference is a multi-phase process involving several individuals and groups. During the submission phase, the authors need to be informed that their papers have been received and they need to be assigned a submission number. Once the submission deadline has passed, the program committee (PC) has to handle the review of the papers, typically by contacting potential referees and asking them to review a number of the papers. Eventually, reviews come in and are used to decide about the acceptance or rejection of the submissions. Authors need to be notified of these decisions and, in case of acceptance, must be asked to produce a revised version of their papers. Finally, 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 organisations, one for each phase of the process. In each organisation, the corresponding MAS can be viewed as being made up of agents associated to the persons involved in the process (authors, PC chair, PC members, reviewers). The roles played by each agent reflect the ones played by the associated person in the conference organisation. They may require agents to interact both directly with each other and indirectly, via an environment composed of papers and review forms. Since an agent is directly associated with a person, and its behaviour can be influenced by that person, opportunistic behaviour can emerge in the application. For example, an author could attempt to review their own paper or a PC member could try to deal with fewer papers than they should. In addition, as the natural environment for the MAS is the Internet — due to the world-wide nature of the conference organisation — interactions with agents external to the MAS itself are likely to occur. For instance, a reviewer may decide to exploit its own personal agent to interact with the other agents of the organisation.

3. Organisational Abstractions

Organisational role models precisely describe all the roles that constitute the computational organisation; in terms of their functionalities, activities, and responsibilities, as well as in terms of their interaction protocols and patterns. Thus the role models establish the position of each role in the organisation [35, 10, 9]. However, role models cannot be considered as the sole organisational abstraction upon which to base the entire development process. Rather, before the design process actually defines the complete role model and, consequently, the whole organisation, a number of other steps need to be performed. Firstly, the analysis phase should identify *how* the organisation is expected to work. Secondly, the design phase should define *which* kind of organisation best fits the requirements identified in the analysis phase. Thirdly, it needs to be determined whether any re-use of available components is possible in some part of the organisational design. When taken together, this necessitates the introduction of three further organisational abstractions: organisational rules (section 3.1), organisational structures (section 3.2), and organisational patterns (section 3.3).

3.1. *Organisational Rules*

The analysis phase aims to collect all the specifications and requirements for building the MAS. To this end, it is possible to identify the basic skills (functionalities and competences) required by the organisation, as well as the basic interactions that are required for the exploitation of these skills. However, until the design phase has decided *which* organisation is most appropriate for the system, the identified skills and interactions cannot fully define the roles and the interaction protocols that will be played in the system. Thus, a complete role model cannot be defined at the analysis phase since this would necessitate a premature commitment to a specific form of organisation. Instead, what the analysis phase should identify are the outline role models and the constraints that the actual organisation, once defined, will have to respect.

The implementation of a computational organisation will have to respect a number of constraints, whose identification can either: *(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/or constraints between roles, protocols, or between roles and protocols. For example, in the case of human organisations: *(i)* social conventions define a set of implicit rules that moderate the interactions between all members (e.g., a clerk cannot contradict or ignore the commands of his manager), *(ii)* company specific conventions might impose constraints on how different roles have to be played (e.g., a clerk cannot take on a role that would require a member of the managing staff to become subordinated). In both of these cases, such global constraints are not best expressed in terms of individual roles or individual interaction protocols. Nevertheless, their identification is important for the correct development of the system and, therefore, they must be taken into account by the designer when actually defining the organisation of the system. To capture this type of information we use the concept of *organisational rules*.

The explicit identification of organisational rules is of particular importance in the context of open systems. With the arrival of new, previously unknown, and possibly self-interested agents, the overall organisation must be able to enforce its internal coherency despite the dynamic and untrustworthy environment. The identification of global organisational rules allows the system designer to explicitly define: *(i)* whether and when to allow new agents to enter the organisation, 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 organisation. In this context, organisational rules may also drive the designer towards the definition of the specific organisation structure that most eases the enforcement of the organisational rules and, for instance, can facilitate the prevention of undesirable behaviour on the part of the unknown agents.

In the manufacturing pipeline example, all the different stages have to maintain the same speed of flow of items in the pipeline. This requirement is best expressed in terms of a global organisational rule, rather than replicating it as a requirement for each and every role in the organisation. In the conference management system, there are a number of rules that drive the proper implementation of the organisation. As notable examples: an agent should be prevented from playing both the role of author and reviewer of the same paper and 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. Instead, they represent global organisational rules.

3.2. *Organisational Structures*

A role model describes all the roles of an organisation and their positions in that organisation. Therefore, a role model also implicitly defines the *topology* of the interaction patterns and the *control regime* of the organisation's activities. That is, it defines the overall *organisational structure*. For example, a role model describing an organisation 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 organisational structure based or a hierarchical topology and on a load partitioning control regime. Other exemplar organisational structures include collectives of peers, multi-level and multi-divisional hierarchies [11], and they can all be modelled in terms of a role model.

However, it is conceptually wrong to think of a role model as something that actually defines the organisational structure. Instead, in the design of a MAS, as well as in the design of any organisation, the role model should derive from the organisational structure that is explicitly chosen. Thus organisational structures should be viewed as first-class abstractions in their own right.

The definition of the system's overall organisational structure can derive from the specifications collected during the analysis phase, as well as from other factors, related to efficiency, simplicity of application design, and organisational theory [11]. In any case, a methodology cannot start the analysis phase by attempting to define a complete role model that implicitly sets the organisational structure. Rather, the definition of the organisational structure is a design choice that should not be anticipated during the analysis phase. In fact:

- starting from the organisational structure — by pretending to know in advance what it should be or by committing a priori to a given organisational structure — may prevent subsequent optimisation and change;
- although the organisational structure of a MAS may be directly driven by its counterpart in the real-world system that the MAS is supposed to support, automate or monitor, this should not automatically imply that the organisation of the software system should mimic that of the real counterpart. Instead, the MAS may be better adopting a different organisational choice. There are several reasons why this could happen:
  - the real world organisation may not be well structured and the analysis phase could highlight several shortcomings;
  - the software, in itself, may change the way of working. Thus, the mere presence of the software introduces changes in the real organisation and these changes need to be reflected in the MAS;
  - the efficiency issues that may have driven a human organisation towards the adoption of a particular organisational structure may not necessarily apply to the agent organisation;

- the organisation, once defined, has to respect the organisational rules. Starting from a pre-defined organisational structure can make it difficult to have the organisational rules respected and enforced. It is more natural for the choice of the organisation structure to follow from the identification of the organisational rules.

In the manufacturing pipeline example, the most natural choice is to have an organisational structure in which all of the stages in the pipeline are peers, and in which they directly interact with their neighbours as needed. For instance, with reference to Figure 2, the stages Stage1, Stage2, Stage3, and Stage4 are controlled by agents R1, R2, R3, and R4, respectively, and each of these agents directly interacts with its neighbours. This closely mimics the structure of the real-world pipeline. However, this is not the only possible choice. Moreover, it may not necessarily be the best one. For instance, because of the real-time nature of the pipeline control problem, it may be the case that a problem that requires global coordination between all the agents cannot be solved in due time, because of the high coordination costs associated with peer-based systems. In such cases, the designer can adopt a different organisational structure: for example, as sketched in Figure 3, a global coordinator agent (RC) can be introduced that is in charge of controlling and mediating the interactions for all the other agents. This, in turn, leads to a hierarchical organisation.

In the conference management example, the overall structure of the organisation can generally be derived from the structure the conference organisers have decided to adopt. However, it is often the case that the same conference varies its organisational structure from year to year, depending on both the size of the conference and the organisers' attitudes. For example, a small conference usually relies solely on the PC members for the review process, and the PC chair acts as a global coordinator, in a single-level hierarchy, for the work of the PC members (see Figure 4). In contrast, a large conference usually has to involve external reviewers. This may require the PC chair to partition the papers among the PC members, and the PC members to be in charge of seeking the appropriate number of reviews for their assigned partition. In other words, the organisational structure is a multi-level hierarchy based on a work partitioning control regime at the highest level (the one of the PC chair) and on a global coordination control regime at the PC member level (see Figure 5).

If the analysis phase commits the system to a specific organisational structure, the designer of the associated MAS will find it difficult to adapt the system, year after year, to the changing needs. For instance, it is very likely that a conference changes its dimensions in different editions and, consequently, its organisational structure. Thus, if the analysis phase simply describes the system's requirements, abstracting away from any specific organisational structure, the designer can reuse it to produce a new design according to the conference's new organisational structure.

3.3. *Organisational Patterns*

There are numerous potential organisational structures, both in terms of topology of the interactions and control regimes [11]. However, we believe that a (comparatively) small subset of these structures are likely to be used most of the time. Thus, only rarely will peculiar structures be adopted (typically when the organisation has a very specific and unusual set of requirements).

Given this observation, we believe there are significant opportunities for re-use. Broadly speaking, any methodology that encourages re-use of pre-defined components and architectures will ease and speed-up the work of both designers and developers. Object-oriented technology has recognised this need and increased the potential for re-use via design-patterns [12]. In this case, the most widely-used patterns of composition and interaction of object-oriented systems have been catalogued, and precisely described in terms of extent of applicability, sample imple-

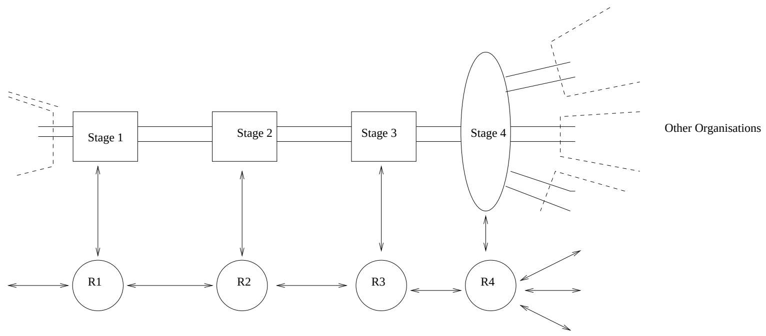

Figure 2: A manifacturing pipeline: pipeline organisation

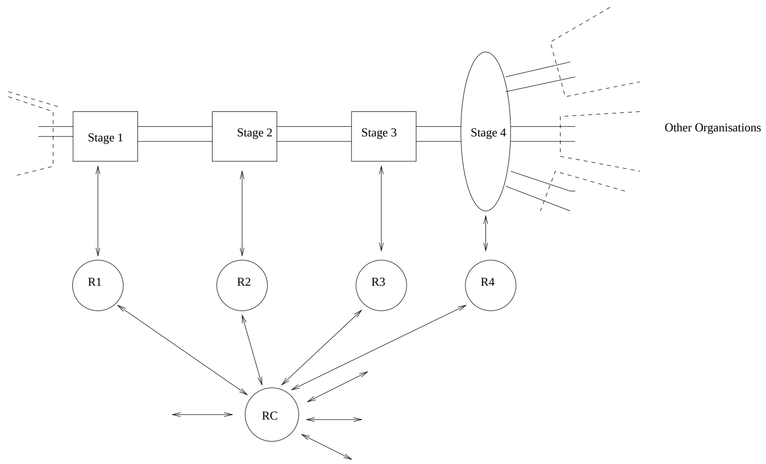

Figure 3: A Manifacturing pipeline: hierarchical organisation

mentation, and use cases. A software designer can then rely on these catalogues, and build applications by composing and re-using not only single objects, but whole pieces of the software architecture.

Turning now to the area of agent-based systems, we envisage something similar with respect to the most widely used organisational structures. Thus with the availability of catalogues of *organisational patterns*, designers can recognise in their MASs the presence of known patterns, and re-use definitions from the catalogue. In addition, designers can also be guided by the catalogue in the choice of the most appropriate organisational patterns for their MAS. Of course, for patterns to be properly exploited, the organisational structure must have been explicitly identified in the design phase.

In the manufacturing example, the pipeline organisation between agents expresses an organisational pattern that is likely to re-appear in many applications (indeed it is one that is already widely exploited as an architectural pattern in traditional software systems [4]). The same can also be said of the hierarchical pipeline structure. In both cases, if a catalogue of patterns was available, the designer could rely on it to help define the system structure.

In the conference management example, the various organisational structures

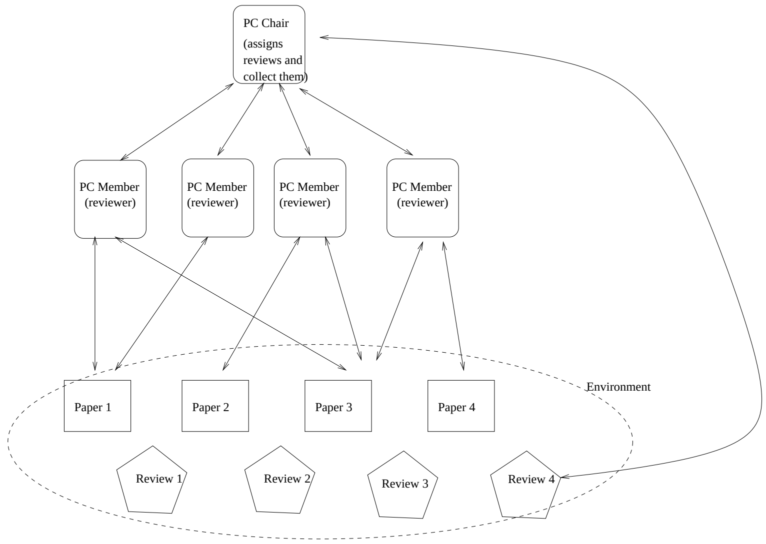

Figure 4: Conference management: In a small conference, the PC chair assigns the reviews directly to PC members and, possibly, to itself

that conferences of different sizes tend to adopt are also all fairly typical: from single hierarchies, to multi-level and divisional ones. Therefore, also in this case, it is expected that a methodology that makes explicit use of organisational patterns would ease the application design.

It is worth mentioning that several attempts to analyse and catalogue organisational agent patterns currently exist [31, 18, 19]. However, in most cases, this work abstracts away from any specific methodology for MAS analysis and design which should encourage and facilitate the re-use of these patterns. This, in turn, makes re-use more difficult.

4. Specifying Organisational Rules

Preliminary roles and protocols capture the basic characteristics, functionalities, and interaction patterns that the MAS has to include, independently of any organisational structure. However, as previously stated, the other characteristics of the system that can be captured during the preliminary analysis phase are the *organisational rules*. These rules play a key role in the analysis of a MAS, in that their correct identification and modelling can drive and strongly influence the subsequent design phase, i.e., the choice of the most suitable organisational structure.

Organisational rules express relationships and constraints between roles, between protocols, and between roles and protocols, that can drive the identification of the organisational structure. To some extent, organisational rules can be considered as *liveness* and the *safety* properties of the organisation, i.e., properties that the system must guarantee to enable "something good" and "nothing bad", respectively, happens in the organisation [23, 24]. Although the concepts of liveness and safety properties have already been exploited – at the intra-component level – in the analysis of concurrent systems, as well as – at the intra-role level – in the analysis of agent-based systems [35], the novelty of the organisational rules abstraction is to introduce analogous concepts at the organisational – inter-agent – level. Still, a formalism that can be used to express intra-role properties can easily be extended

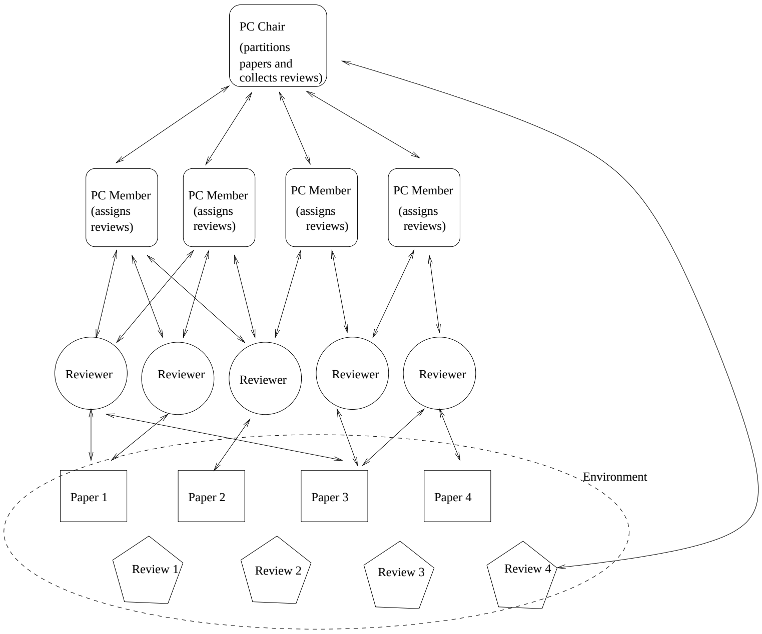

Figure 5: Conference management: In a large conference, the PC chair partitions the papers among the PC members that, in turn, are in charge of finding the appropriate referees for their assigned papers and of collecting the reviews

to express organisational rules.

In the remainder of this section, we introduce a formal notation based on first-order temporal logic for expressing organisational rules. Then we show, via several application examples, how the formalism can be used in the analysis of a MAS. Clearly, the availability of a suitable formalism to express organizational rules can help both checking possible inconsistencies in analysis and verifying that the MAS design respect the analysis specification. However, detailing how this can actually be done is outside of the scope of this paper.

4.1. *Organisational Rules and Temporal Logic*

As shown by the informal examples introduced in subsection 3.1, the relations and constraints (or, from a different perspective, the liveness and safety properties) that one may wish to express as organisational rules may be instantaneous (i.e., referring to some particular moment in time), or *temporal*, i.e., relating roles and protocols over some period of time. Because of their temporal characteristics, a formalism explicitly introducing a notion of time, as temporal logic does, is clearly suited to expressing organisational rules. Indeed this has already been proved by the fact that temporal logic has been adopted to express safety and liveness properties at the intra-agent level [35]. However, before presenting the notation, we clarify our choice by giving some informal examples of the kinds of relationships and constraints that we may wish to express.

First, we can express constraints on which agents can play which roles. For example:

- No agent can simultaneously play roles r_1 and r_2.

Formula	Interpretation
$\bigcirc\varphi$	φ is true next
$\Diamond\varphi$	φ is eventually true
$\Box\varphi$	φ is always true
$\varphi\,\mathcal{U}\,\psi$	φ is true until ψ is true
$\varphi\,\mathcal{W}\,\psi$	φ is true unless ψ is true
$\varphi\,\mathcal{B}\,\psi$	φ is true before ψ is true

Table 1: Temporal connectives.

- Every agent that plays role r_1 must play at least one other role.

Other constraints relate to the *cardinality* of roles. For example:

- At most 5 agents can play role r_1.
- Between 1 and 10 agents can play role r_1.

Finally, we can consider constraints that relate to the temporal dynamics of roles:

- Every agent that plays role r_1 will eventually play role r_2.
- Some agent plays role r_1 until some other agent plays role r_2.
- Once an agent plays role r_1, it plays r_1 forever.

Similar constraints may also apply to protocols, expressing how and when protocols may be (can be) executed. In order to express such rules, we use a subset of first-order temporal logic.* An exposition of the syntax, semantics, and proof theory of first-order temporal logic is beyond the scope of this paper. Thus we concentrate on giving an informal introduction to the logic and its use. The basic idea is to extend classical first-order logic by introducing a collection of temporal operators, summarised in table 1. These connectives have the following interpretation:

- $\bigcirc$ means "next" — thus $\bigcirc\varphi$ is satisfied now if φ is satisfied at the next moment;
- $\Diamond$ means "sometime" — thus $\Diamond\varphi$ is satisfied now if φ is satisfied either now or at some future moment;
- $\Box$ means "always" — thus $\Box\varphi$ is satisfied now if φ is satisfied now and at all future moments;
- $\mathcal{U}$ means "until" — thus $\varphi\,\mathcal{U}\,\psi$ is satisfied now if ψ is satisfied at some future moment, and φ is satisfied until then — $\mathcal{W}$ is a binary connective similar to $\mathcal{U}$, allowing for the possibility that the second argument might never be satisfied.
- $\mathcal{B}$ means "before" — thus $\varphi\,\mathcal{B}\,\psi$ means that ψ is eventually true, and at some time before this, φ is true.

*There are many different kinds of temporal logic; the particular one we make use of is that pioneered by Manna, Pnueli, and colleagues for specifying and verifying the properties of reactive systems [23, 24].

To illustrate the use of these temporal connectives, consider the following examples:

$$\Box\, important(agents)$$

means "it is now, and will always be true that agents are important".

$$\Diamond\, important(Gaia)$$

means "sometime in the future, Gaia will be important"

$$(\neg friends(us))\,\mathcal{U}\; apologise(you)$$

means "we are not friends until you apologise". And finally,

$$\bigcirc\, apologise(you)$$

means "tomorrow (in the next state), you apologise". This final example may need some explanation: the temporal model that underpins the logic is based on a model of time which consists of a linear discrete sequence of states. Using the "$\bigcirc$" operator makes it possible to say that something is true in the next state in this sequence.

In order to express organisational rules using first order temporal logic, we make use of some predicates and functions that have special meanings. The first of these is $plays(\ldots)$. This is a two-place predicate, in which the first term stands for an agent and the second term stands for a role. The predicate $plays(i, r)$ means that the agent denoted by i plays the role denoted by r. Second, we make use of a function $card(\ldots)$ which takes a single argument — a role — and returns the number of agents that currently play this role ("card" is an abbreviation for "cardinality"). Terms that stand for roles are also functions; the arguments they take are the parameters that are supplied to them in the role definition.

To illustrate the use of this simple notation for expressing organisational rules, we return to the following examples:

- No agent can simultaneously play roles r_1 and r_2.

$$\forall i \cdot \neg[Plays(i, r_1) \wedge Plays(i, r_2)]$$

- Every agent that plays role r_1 must play at least one other role.

$$\forall i \cdot Plays(i, r_1) \Rightarrow \exists r_2 \cdot Plays(i, r_2) \wedge r_1 \neq r_2$$

- At most 5 agents can play role r_1.

$$card(r_1) \leq 5$$

- Between 1 and 10 agents can play role r_1.

$$1 \leq card(r_1) \leq 10$$

- Every agent that plays role r_1 will eventually play role r_2.

$$\forall i \cdot Plays(i, r_1) \Rightarrow \Diamond Plays(i, r_2)$$

  Notice that this is clearly a *liveness* property of the organisation, stating that "something good happens".

- Some agent plays role r_1 until some other agent plays role r_2.

$$\exists i \cdot \mathit{Plays}(i, r_1)\, \mathcal{U}\, \exists j \cdot \mathit{Plays}(j, r_2)$$

- Once an agent plays role r_1, it plays r_1 forever.

$$\forall i \cdot \mathit{Plays}(i, r_1) \Rightarrow \Box \mathit{Plays}(i, r_1)$$

Notice that this is clearly a *safety* property of the organisation, stating that "nothing bad happens".

4.2. *Example Organisational Rule Specifications*

To clarify the use of the above introduced notation, let us refer again to the pipeline and the conference management examples.

Manufacturing Pipeline: In the manufacturing pipeline, inconsistency and conflict avoidance require that only one agent can play the role of controller for a given stage of the pipeline. This condition is a first, basic, organisational rule of the system, and it can be expressed as follows:

$$\forall s \cdot \forall i, j \cdot \mathit{Plays}(i, \mathit{controller}(s)) \wedge \mathit{Plays}(j, \mathit{controller}(s)) \Rightarrow (i = j)$$

where $\mathit{controller}(s)$ denotes the role of controller for stage s of the pipeline.

As a second example, one of the major responsibilities of an agent playing the role of stage controller is to guarantee to correctly process all the items flowing in the pipeline, without blocking the pipeline itself. This implies that, at any time, the flow of items must be the same for each stage of the pipeline. Since this safety condition must be guaranteed by all the stages, it is most appropriate to specify it as a global organisational rule:

$$\forall s, s' \cdot \mathit{flow}(s) = \mathit{flow}(s') \wedge \mathit{flow}(s) = \mathit{agreedFlow}$$

where $\mathit{flow}(s)$ is the estimated or requested flow of items in stage s, as identified by the analysis phase, and *agreedFlow* is the flow that has been agreed by the stage controllers.

In this scenario, one of the stage controllers may detect that the actual flow of items in the pipeline can no longer be sustained, for example because of changed conditions. In such cases, the stage controller agent cannot simply change its local flow of items. Instead, if the flow of items has to be changed in one stage, it has to be changed in the whole pipeline. Thus, if one of the stages discovers that it is unable to guarantee a given flow, it must immediately initiate a protocol to request a change in the global flow.

To express organisational rules related to the execution of protocols, we make use of two binary predicates: $\mathit{initiate}(r, p)$ and $\mathit{participate}(r, p)$, (where r is a term denoting a role and p is a term denoting a protocol), expressing that role r initiates or participates in, respectively, the execution of protocol p.

The fact that a stage must initiate the protocol *RequestChangeFlow* to request a change in flow can be expressed by the following organisational rule:

$$\begin{array}{l} \forall s \cdot \mathit{flow}(s) \neq \mathit{agreedFlow} \Rightarrow \\ \bigcirc\, \mathit{initiate}(\mathit{controller}(s), \mathit{RequestChangeFlow}(\mathit{flow}(s))) \end{array}$$

Using a similar notation, we can express the additional rule stating that, if a stage initiates a *RequestChangeFlow* protocol, all other stages must eventually be involved in the same protocol:

$$\exists s \cdot initiate(controller(s), RequestChangeFlow(newflow)) \Rightarrow \\ \forall s' \cdot \Diamond participate(controller(s'), RequestChangeFlow(newFlow))$$

Complimentary to the above conditions, (which can be considered liveness properties), we have a safety condition expressing the fact that none of the stages can actually change their flow without having involved all other stages in a *RequestChangeFlow* protocol. This can be expressed by the following organisational rule:

$$\forall s \cdot participate(controller(j), RequestChangeFlow(newflow))\,\mathcal{B} \\ participate(controller(s), ChangeFlow(newflow))$$

It is important to note that the aforementioned organisational rules do not commit the designer to any specific organisational structures. They simply state what an organisational structure should guarantee for the pipeline to work correctly. For instance, the fact that, if one stage initiates a *RequestChangeFlow* protocol, all other stages must be eventually involved in the same protocol, does not automatically imply that it is the stage that has initiated the protocol that has to involve all other ones in it. Neither does it imply that there is a global controller in charge of involving all stages in the protocol. The organisational rules must leave both choices open, and must simply state that there is a need for global coordination between roles. It is then up to the designer to decide whether to introduce a global controller to be in charge of handling all problems and protocols related to the flux changes, or to let each stage directly involve the other stages in the necessary protocols.

Of course, any real-world manufacturing pipeline is likely to have further global constraints placed upon its operation and these will require additional, more specific organisational rules to be specified.

Conference Management: As stated in subsection 3.1, organisational rules are very useful for preventing and controlling opportunistic and self-interested behaviour. Thus, the power of organisational rules becomes even more apparent when applied to the conference management example. Here conflicts and self-interest may influence interactions, and there are more complex relationships between roles and between roles and protocols.

In general, the conference's review process must guarantee that every paper is reviewed by at least three reviewers. This simple and basic rule relates to the cardinality of roles, i.e., the number of times that a given role must be played, and it can be expressed as follows:

$$\forall p \cdot card(reviewer(p)) \geq 3$$

However, during the review process, it must also be guaranteed that a single reviewer does not receive the same paper for review multiple times. This can be expressed by the following organisational rule:

$$\forall i \cdot \forall p \cdot Plays(i, reviewer(p)) \Rightarrow \bigcirc \Box \neg Plays(i, reviewer(p))$$

stating that, if one agent i plays the role of reviewer for both a paper p then it does not play this role subsequently. This is a safety property.

The above organisational rule may seem trivial, in that in a real-world conference managed directly by humans such a condition is often implicitly assumed. However, if the review process is automatically managed by agents, the explicit definition of such a rule during the analysis phase is necessary for the correct implementation of the organisation. Also, since an opportunistic agent may have an interest in reviewing the same paper several times (e.g., because the agent is the personal assistant of a reviewer who wants to strongly influence the destiny of that paper), the explicit definition of the organisational rule makes it possible for the design to be tuned so as to prevent the breaking of such rules.

In this scenario, further specific conditions must be guaranteed on reviews to ensure the review process is fair and one in which possibly self-interested behaviour can be controlled. As another example, we can express the fact that no agent is both the reviewer and the author of a paper:

$$\forall i \cdot \forall p \cdot \mathit{Plays}(i, \mathit{author}(p)) \Rightarrow \Box \neg \mathit{Plays}(i, \mathit{reviewer}(p))$$

A similar condition must apply to PC members (or PC chairs). In fact, given that a PC chair or a PC member is in charge of collecting reviews for a given (sub)set of papers, one must avoid a PC member selecting reviewers and collecting reviews for a paper of which it is the author:

$$\forall i \cdot \forall p \cdot \mathit{Plays}(i, \mathit{author}(p)) \Rightarrow \Box \neg \mathit{Plays}(i, \mathit{review\text{-}collector}(p))$$

Rules related to the execution of protocols also have to be imposed on the organisation. For example, we need to specify that once a reviewer has received a paper to review, the reviewer must eventually send in a review of that paper. By making use of the protocol-related predicates introduce in the manufacturing example, we can express this organisational rule as follow:

$$\forall i \cdot \forall p \cdot \mathit{participate}(i, \mathit{receivePaper}(p)) \Rightarrow \Diamond \mathit{inititate}(i, \mathit{submitReview}(p))$$

This is an example of a liveness property related to the correct execution of protocols. We can also specify the corresponding safety property, i.e., that no review is submitted unless the reviewer has previously received that paper to review.

$$\forall i \cdot \forall p \cdot \mathit{participate}(i, \mathit{receivePaper}(p))\, \mathcal{B}\, \mathit{initiate}(i, \mathit{submitReview}(p))$$

Finally, before the program chair can collect all the reviews and finalise the list of accepted papers, it is necessary to ensure that a sufficient number of reviews have been submitted. For example, that, for each and every paper, at least two reviews have been received for a paper the chair initiates a decision on this paper:

$$\forall p \cdot [\mathit{submittedReviews}(p) > 2]\, \mathcal{B}\, \mathit{initiate}(\mathit{chair}, \mathit{decision}(p))$$

As in the case of the manufacturing pipeline, it is likely that a given conference would identify further, more specific, organisational rules. Again, we emphasise the fact that organisational rules can and have to be identified and expressed without making any assumption on the actual organisational structure.

5. Towards an Organisation-Oriented Methodology

Having introduced and defined the various organisational abstractions that we believe are necessary for analysing and designing open agent systems, 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. Figures 6 sketches this process – which we consider as an extension of the Gaia one, the details of which are given in [35] – and outlines the central role played by the introduced organisational abstractions. In what follows,

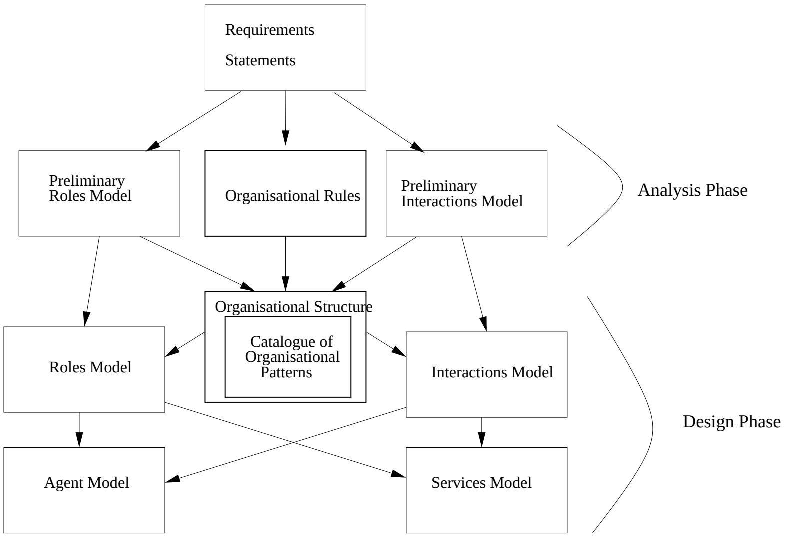

Figure 6: Models of the Methodology

we only give an outline of the process – a detailed exposition is beyond the scope of this paper.

This work does not deal with the activities of requirements capture, since this is an endeavour that is neutral with respect to the eventual solution technology. Thus, we start with the analysis phase. This phase is tasked with collecting all the specifications from which the design of the computational organisation can start. This includes the identification of:

- the overall goals of the organisation and its expected global behaviour;
- the basic skills required by the organisation and the basic interactions required for the exploitation of these skills (that is, preliminary role and interactions models);
- the rules that the organisation should respect and/or enforce in its global behaviour.

The output of the analysis phase consists of three basic models: (i) a preliminary roles model; (ii) a preliminary interactions model; and (iii) a set of organisational rules. The roles model is derived from the list of the organisation's basic skills. It contains only those roles that can be identified without committing to the imposition of a specific organisational structure. Here we assume the Gaia representation of roles; that is, a role is defined in terms of: responsibilities (the functionality of the role expressed in terms of safety and liveness properties), permissions (the set of rights associated with the role), activities (computations associated with the role that may be carried out by the agent without interacting with other agents), and protocols (ways in which the role can interact with other roles). The preliminary interactions model captures the protocols that are associated with the preliminary roles. Again we assume that this model uses the same structure as Gaia. Thus a protocol is defined in terms of its purpose (a brief textual description of the nature of the interaction), initiator (the role(s) responsible for starting the interaction), responder (the role(s) with which the initiator interacts), inputs (information used by the role initiator while enacting the protocol), outputs (information supplied by/to the protocol responder during the course of the interaction) and processing (brief textual description of any processing the protocol initiator performs during the course of the interaction). Both the preliminary role and interaction models will be refined and expanded upon in the subsequent design phase. The final analysis model captures the global organisational rules (as discussed in section 4).

The design phase builds on the output of the analysis phase and produces a complete specification of the MAS. To this end, design can be decomposed into the following phases:

- definition of the organisational structure; by choosing the topology and the control regime. This involves considering: *(i)* the overall organisational efficiency, *(ii)* the corresponding (if any) real-world organisation in which the MAS is situated, and *(iii)* the need to respect and enforce the organisational rules.
- completion of the preliminary role and interaction models; based upon the adopted organisational structure, and by keeping the organisational-independent aspects (detected from the analysis phase) and the organisational-dependent ones (deriving from the adoption of a specific organisational structure and from the insertion of roles and protocols in it) as separate as possible;
- exploitation of well-known organisational patterns in the design of the organisational structure and in the design of the final interactions model.
- definition of the agent model (as in Gaia). This identifies the *agent types* that will make up the system, and the *agent instances* that will be instantiated from these types. Here an agent type can best be thought of as a set of agent roles. There may in fact be a one-to-one correspondence between roles and agent types. However, this need not be the case. A designer can choose to package a number of closely related roles in the same agent type for the purposes of convenience.
- definition of the services model (as in Gaia). This identifies the main services that are required to realise the agent's role. A service is simply a single, coherent block of activity in which an agent will engage. For each service that may be performed by an agent, it is necessary to document its properties. Specifically, we must identify its inputs, outputs, pre-conditions, and post-conditions.

As in Gaia, we view the output of the design phase as a specification that can be picked up by using a traditional method (such as object orientation or componentware) or that could be implemented using an appropriate agent-programming framework should one be available. Thus the mapping of the generic design models into a particular implementation architecture is not dealt with in our work.

6. Related Work

Traditional analysis and design methodologies, such as object-oriented ones [2], are poorly suited to MASs because of the fundamental mismatch between the abstractions they provide [35]. Consequently, we believe that those efforts that attempt to simply extend object-oriented methodologies to MAS [20, 18] will inevitably fall short. Moreover, traditional compositional methods for object-oriented software architectures [28, 3] also have limited applicability in the definition of organisations for MASs. Firstly, the defined interaction models are too static when compared to the dynamic interaction models needed by agents. Secondly, the functionality-oriented modelling of the interactions between the system components clashes with the role-oriented perspective of MASs.

A number of agent-specific modelling techniques and development methodologies have been proposed in recent years (see [15] for a survey), several of which attempt to exploit the idea of a MAS as a computational organisation. In most cases, these organisation-oriented systems and modelling techniques define an organisation simply as a collection of roles (i.e., a role model). This is precisely what happens, for example, in the ALAADDIN system [10] where "the group structure" is simply the collection of roles that compose the organisation. Analogously, in the ToolKit approach [9], an organisation 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 organisational rules or organisational structures and, for the reasons we have outlined, will be limited in the range of agent systems they can deal with. In addition, these proposals do not attempt to define a complete and clear methodology for the development of agent system organisations.

Gaia starts from the organisational metaphor and defines a complete methodology [35] for the development of multi-agent systems. It also provides a clean separation between the analysis and design phases. However, it suffers from several limitations that are caused by the incompleteness of its organisational abstractions. The objective of Gaia's analysis phase is 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 assumes that the overall organisational structure is known a priori. However, as already stated, this is not always the case. In addition, by focusing exclusively on the role model, the analysis phase fails to identify any global organisational rules (making Gaia unsuitable for modeling open systems and for controling the behaviour of self-interested agents). Similar shortcomings also effect most of the other organisation-oriented methodologies. For example, the MASE methodology [33] provides clean guidelines for developing multi-agent systems, based on a well-defined six-step process. This process drives developers from analysis to implementation. However, once again, the design process fails to identify any organisational abstraction other than the role model.

From a different perspective, some work in the area of coordination models and languages [13, 7, 6] does explicitly address the problem of defining global rules ("coordination laws") to specify the behaviour and the interaction of agent ensembles. In this work, all interactions have to occur via specific "coordination media", whose internal behaviour can be programmed so as to implement specific policies for governing agent interactions. However, only recently have coordination models been recognised as useful abstractions upon which to define methodologies for the analysis and design of open agent systems. To achieve this, the coordination media are exploited as both the conceptual and physical repository of the organisational rules [8, 30, 29]. A somewhat similar approach has driven the implementation of the Fishmarket system for agent-mediated auctions [27]. In the Fishmarket, the need to force agents to act in accordance with the "social conventions" that rule the organisation of an auction is recognised. To enact social conventions, the system dynamically associates a "controller agent" with each agent in the auction. Controller agents act as a coordination media, in charge of mediating all the interactions and of making agents respect the auction's conventions.

7. Conclusions and Future Work

This paper has discussed a number of issues related to the analysis and design of multi-agent systems. Specifically, we have considered the view of developing multi-agent systems as a process of constructing computational organisations. To date, the organisational concept of role models has become an important research area in the field of agent-based systems. However in this paper we have introduced three further organisational abstractions: organisational rules, organisational structures, and organisational patterns. These concepts, although neglected by the current methodologies for agent-oriented software engineering, are nevertheless of fundamental importance in multi-agent systems, and we therefore believe they should play a central role in any methodology that aims to have broad appeal. Having introduced and motivated these organisational abstractions, we then focused on the concept of organisational rules and showed that a formalism based on temporal logic is well suited to specifying them. Finally, we have sketched some general guidelines for a new methodology for the analysis and design of multi-agent systems that is centered around organisational abstractions.

Further work is needed to detail the proposed methodology, by:

- completing the formalisation of the introduced organisational abstractions, to couple the temporal logic formalism for organisational rules with a suitable formalism to express and analyse organisational structures;
- providing suitable notations for expressing the expected outputs of the analysis and design phases. Here we expect standard notations, such as UML, to be rapidly adapted to the needs of agent-based software engineering [1], as well as the development of new agent-specific models;
- identifying guidelines that assist the designer in the identification of suitable organisational structures for the system. Here analytical methods, experimental results, and case study experiences are likely to be helpful in supporting the choice.

For all of the above topics, we expect significant cross-fertilisation of models, formalisms and experiences from a number of different research areas. Among others, the research area of requirements engineering [26] has several relations with our concept of organisational rules that appear worthy of investigation. In particular, it might provide further useful guidelines and formalisms for the identification and the modelling, respectively, of organisational rules. In addition, the research results of coordination, organisational and management sciences [22, 32, 25], which have widely studied the structures of human organisations and their most common patterns, are expected to play a significant role in guiding the choice of organisational structure during the design phase.

8. References

1. B. Bauer, J. P. Muller, and J. Odell. Agent uml: A formalism for specifying multiagent software systems. 2001.
2. G. Booch. *Object-oriented Analysis and Design (second edition).* Addison Wesley, Reading (MA), 1994.
3. F.M.T. Brazier, B.M. Dunin-Keplicz, N.R. Jennings, and J. Treur. Desire: Modelling multi-agent systems in a compositional formal framework. *Journal of Cooperative Information Systems*, 6(1):67–94, 1997.
4. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stahl. *A System of Patterns.* Wiley, 1998.
5. S. Bussmann. Agent-oriented programming of manifacturing control tasks. In *Proceeding of the 3rd International Conference on Multi-Agent Systems (ICMAS 98)*, pages 57–63. IEEE CS Press, June 1998.
6. G. Cabri, L. Leonardi, and F. Zambonelli. Mars: a programmable coordination architecture for mobile agents. *IEEE Internet Computing*, 4(4):26–35, July-Aug. 2000.
7. P. Ciancarini. Coordination models and languages as software integrators. *ACM Computing Surveys*, 28(2), June 1996.
8. P. Ciancarini, A. Omicini, and F. Zambonelli. Multiagent systems engineering: the coordination viewpoint. In *Intelligents Agents VI (ATAL99)*, volume 1767 of *LNAI*, pages 250–259. Springer Verlag, 2000.

9. Y. Demazeau and A. C. Rocha Costa. Populations and organizations in open multi-agent systems. In *1st National Symposium on Parallel and Distributed AI (PDAI'96)*. 1996.
10. J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceeding of the 3rd International Conference on Multi-Agent Systems (ICMAS 98)*. IEEE CS Press, June 1998.
11. M. S. Fox. An organizational view of distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):70–80, January 1981.
12. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison Wesley, Reading (MA), 1995.
13. D. Gelernter and N. Carriero. Coordination languages and their significance. *Communications of the ACM*, 35(2):97–107, February 1992.
14. M. H. Huhns. Interaction-oriented programming. In *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering*, volume 1957 of *LNCS*. Springer Verlag, 2000.
15. C. Iglesias, M. Garijo, and J. Gonzales. A survey of agent-oriented methodologies. In A. S. Rao J.P. Muller, M. P. Singh, editor, *Intelligents Agents IV (ATAL98)*, LNAI. Springer-Verlag, 1999.
16. N. R. Jennings. Agent-based computing: Promises and perils. In *International Joint Conference on Artificial Intelligence (IJCAI 99)*, pages 1429–1436, 1999.
17. N.R. Jennings. On agent-based software engineering. In *Artificial Intelligence*, volume 117, pages 277–296. 2000.
18. E. A. Kendall. Role modelling for agent system analysis, design, and implementation. In *1st International Symposium on Agent Systems and Applications*. IEEE CS Press, October 1999.
19. E. A. Kendall. Agent software engineering with role modelling. In *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering*, volume 1957 of *LNCS*. Springer Verlag, 2000.
20. D. Kinny and M. Georgeff. A methodology and modelling technique for systems of bdi agents. In *Workshop on Modelling Autonomous Agents in a Multi-Agent World, LNAI 1038*, pages 56–71. Springer-Verlag, 1996.
21. J. Lind. Issues in agent-oriented software engineering. In *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering*, volume 1957 of *LNCS*. Springer Verlag, 2000.
22. T. W. Malone and K. Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119, March 1994.
23. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer Verlag, 1992.
24. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems — Safety*. Springer Verlag, 1995.
25. H. Mintzberg. *The Structuring of Organizations: A Synthesis of the Research*. Prentice Hall, Englewood Cliffs, N.J., 1979.
26. J. Mylopoulos, L. Chung, and B. Nixon. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Transactions on Software Engineering*, 18(6):483–497, June 1992.
27. P. Noriega. *Agent-mediated Auctions: The Fishmarket Metaphor*. Ph.D Thesis, Universitat Autonoma de Barcelona, Barcelona (E), 1997.
28. J. Odell, H. Van Dyke Parunak, and C. Bock. Representing agent interaction protocols in uml. In *OMG Document ad/99-12-01*. Intellicorp Inc., December 1999.
29. A. Omicini. Soda: Societies and infrastructures in the analysis and design of agent-based systems. In *Proceedings of the 1st International Workshop on Agent-Oriented

Software Engineering, volume 1957 of *LNCS*. Springer Verlag, 2000.
30. A. Omicini and F. Zambonelli. Coordination for Internet application development. *Journal of Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, 1999.
31. Y. Tahara, A. Ohsuga, and S. Honiden. Agent system development based on agent patterns. In *International Conference on Software Engineering*, pages 356–367. ACM, 1999.
32. J.D. Thompson. *Organizations in Action*. McGraw-Hill, New York, 1967.
33. M. Wood, S. A. DeLoach, and C. Sparkman. Multiagent system engineering. *International Journal of Software Engineering and Knowledge Engineering*, 2001.
34. M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.
35. M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.
36. M.J. Wooldridge. Agent-based software engineering. *IEE Proceedings on Software Engineering*, 144(1):26–37, February 1997.
37. F. Zambonelli, N. Jennings, A. Omicini, and M. Wooldridge. Agent-oriented software engineering for internet applications. In A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, editors, *Coordination of Internet Agents: Models, Technologies and Applications*. Springer Verlag, 2001.