Use the following reference for this paper:
Agent-Oriented Software Engineering (AOSE) IV, P. Giorgini, Jörg Müller, James Odell, eds, Melbourne, July 2003, LNCS 2935, pp. 214-230, 2004

From Agents to Organizations: an Organizational View of Multi-Agent Systems

Jacques Ferber[1], Olivier Gutknecht[1], and Fabien Michel[1]

[1] LIRMM – University of Montpellier II, 161 rue Ada,
34592 Cedex 5, Montpellier, France
69042 Heidelberg, Germany
{ferber, olg, fmichel}@lirmm.fr

Abstract. While multi-agent systems seem to provide a good basis for building complex software systems, this paper points out some of the drawbacks of classical "agent centered" multi-agent systems. To resolve these difficulties we claim that organization centered multi-agent system, or OCMAS for short, may be used. We propose a set of general principles from which true OCMAS may be designed. One of these principles is not to assume anything about the cognitive capabilities of agents. In order to show how OCMAS models may be designed, we propose a very concise and minimal OCMAS model called AGR, for Agent/Group/Role. We propose a set of notations and a methodological framework to help the designer to build MAS using AGR. We then show that it is possible to design multi-agent systems using only OCMAS models.

1 Introduction

Since their coming out in the 80's multi-agent systems have been considered as "societies of agents", i.e. as a set of agents that interact together to coordinate their behavior and often cooperate to achieve some collective goal. It is clear, from this conception, that the body of multi-agent researches should be concerned by both agents and societies. However, an important emphasis has been put on the agent side. Multi-agent systems have particularly been studied at the micro-level, i.e. at the level of the states of an agent and of the relation between these states and its overall behavior. In this view, communications are seen as speech acts whose meaning may be described in terms of the mental states of an agent. The development of communication languages such as KQML and FIPA ACL follows directly from this frame of mind.

We will use the term "agent centered multi-agent system" or ACMAS for short to talk about this type of classical multi-agent systems designed in terms of agents' mental states. As we will see in the following section, ACMAS suffer from some weaknesses that cannot be solved at the agent level, because they reside deep in the core of ACMAS foundational principles.

Recently a particular interest has been given to the use of organizational concepts within MAS where the concepts of 'organizations', 'groups', 'communities', 'roles',

Paper presented at AOSE'2003 (Agent Oriented Software Engineering), Melbourne July 2003, to be published in LNCS, Springer Verlag, 2003.

'functions', etc. play an important role [9] [16] [14] [4] [13]. We will call 'organization centered multi-agent systems' or OCMAS for short, multi-agent systems whose foundation lie in this kind of organizational concepts.

Thinking in terms of organization design differs from the agent-centered approach that has been dominant during many years. An organization oriented MAS is not considered any more in terms of mental states, but only on capabilities and constraints, on organizational concepts such as roles (or function, or position), groups (or communities), tasks (or activities) and interaction protocols (or dialogue structure), thus on what relates the structure of an organization to the externally observable behavior of its agents. However, while OCMAS might solve, as we will see, the main weaknesses of ACMAS, their characteristics and consequences, have somehow been left out and have not been presented clearly. We will see in this paper, that it is possible to design MAS using only organizational concepts. At first, this approach needs a new state of mind to get away from the agent oriented, now classical, conception. However, it does not mean that agent mental states must be thrown away; we only want to stress that it is possible to build organizations as frameworks where agents with different cognitive abilities may interact.

Section 2 will show that some of the weaknesses of ACMAS appear as consequences of the mere foundational principles, somehow implicit, of ACMAS. Section 3 will introduce the main concepts of OCMAS and a set of fundamental principles that could be considered as a kind of manifesto for designing MAS from a pure organizational perspective.

In order to show that it is possible to design OCMAS in this framework, we will present, in section 4, a generic but simple organizational model for building OCMAS, called AGR for Agent/Group/Role. This presentation will include the basic concepts and the notation one can use to describe organizations. The remaining sections will introduce a simple example and a sketch of a methodology based on these organizational concepts.

2 Drawbacks of ACMAS

1.1 Analysis of some drawbacks of the classical ACMAS approach

It has been shown that the world of software engineering may benefits from the concepts and architectures proposed by the MAS community [9, 16] in order to simplify the design of complex software systems.

In order to make MAS systems ready for industrial applications, a non-profit association called FIPA, has proposed a set of norms and standards that designers of multi-agent systems should meet to make their MAS compatible with other systems. An interesting point about these standards, and the platforms and the platform that have been built according to them (see Jade and Fipa-OS for instance), is that they are based on some assumption that lies somewhere in the core of most of early work on MAS.

1. An agent may communicate with any other agent

2. An agent provides a set of services, which are available to every other agent in the system.
3. It is the responsibility of each agent to constrain its accessibility from other agents.
4. It is the responsibility of each agent to define its relation, contracts, etc. with other agents. Thus, an agent "knows" directly (through its acquaintances) the set of agents with which it may interact.
5. Each agent contains with its name its way to be accessed from the outside (the notion of Agent ID well known by all designers of MAS). Therefore, agents are supposed to be autonomous and no constraint is placed on the way they interact.

In this situation, as Jennings and Wooldridge have been pointed out, ACMAS may suffer some drawbacks when engineering large systems:

"Another common misconception is that agent-based systems require no real structure. While this may be true in certain cases, most agent systems require considerably more system-level engineering than this. Some way of structuring the society is typically needed to reduce the system's complexity, to increase the system's efficiency, and to more accurately model the problem being tackled." [10]. This leads to two major drawbacks, according to Jennings [9]:

- The patterns and the outcomes of the interactions are inherently unpredictable.
- Predicting the behavior of the overall system based on its constituent components is extremely difficult (sometimes impossible) because of the high likelihood of emergent (and unwanted) behavior.

Surely, freedom has a price: it is not possible to suppose that agents designed by different designers could interact altogether without any problems. Some assumptions have to be made about the primitives of communications (the "performatives" of the language) and about the architecture of agents (for instance, agents may be assumed to behave purposively in a cognitive way, using some kind of BDI architecture). However, agents do not have access to these constraints that are specified as ISO-like standards, and they do not have the possibility to accept, or refuse, to follow them. This imposes a strong homogeneity on agents: agents are supposed to use the same language and to be built using very similar architectures. The other weaknesses of these MAS are:

1. **Security of applications**: The possibility that all agents may communicate without any external control may lead to security problems. When all agent may interact freely altogether, it is the responsibility of agents (and therefore of the application designer) to check the qualification of its interlocutors} and to implement security controls. Because there is no "general" security management, it is easy for an agent to act as a pirate and use the system fraudulently. On the contrary, too strong security measures could prevent the system to work efficiently on domains where speed and response is more important than security.
2. **Modularity**: in classical software engineering, entities that closely work together are grouped into modules or "packages". For each module, rules of visibil-

ity are defined. Some entities may be seen by other packages (and even by the whole software) whereas others, so called *private* entities, are hidden and therefore not accessible from outside the package. This is not possible with AOMAS where all agents are accessible from everywhere. It should be important to propose a way to group together agents that have to work together. However, this proposal should not stay on static grounds, but propose a way to group together active agents that work together. Moreover, agents should be able to modify dynamically their grouping during their lifetime, according to some general design rules.

3. **Framework/component approach**. Modern software engineering has shown the importance of the framework/component concept. A framework is an abstract architecture in which components plug-in. It is often necessary to define sub-frameworks of frameworks. For instance in a GUI framework, some heavy components, such as charts, trees or tables (see the JTree or JTable components of Java), may introduce their own sub-framework. Unfortunately, in ACMAS, there is only one framework, the platform itself, and it is not possible to describe sub-framework in which specific interactions could be built.

1.2 Solution

To overcome these difficulties, Jennings proposes a solution in the definition of a social level characterization of agent based systems, which follows Newell's levels of computer systems. However, this paper did not develop the main features of organization and their consequences in the process of analysis and design of MAS.

In the following, we will extend and continue these prospects by presenting and analyzing the main concepts of organization centered multi-agent systems (OCMAS) and their properties for building MAS. During our discussion, we will focus on a specific model of OCMAS, called AGR, for Agent/Group/Role, a simple though very powerful and generic organizational model of multi-agent systems.

3 Organization centered MAS

3.1 Definitions

There are several definitions of what an organization exactly means. Indeed, the word "organization" is a complex word that has several meanings. In [6], Gasser proposed the definition of organization to which we subscribe:

An organization provides a framework for activity and interaction through the definition of roles, behavioral expectations and authority relationships (e. g. control).

This definition is rather general and does not provide any clue on *how* to design organizations. In [15] Jennings and Wooldridge propose a more practical definition:

We view an organisation as a collection of roles, that stand in certain relationships to one another, and that take part in systematic institutionalised patterns of interactions with other roles".

However, this definition lacks a very important feature of organizations: their partitioning, the way boundaries are placed between sub-organizations. Except in very small organizations, organizations are structured as aggregates of several partitions, sometimes called groups or communities, contexts, department, services, etc. and each partition may itself be decomposed into sub-partitions. From these definitions, it is possible to derive the main features of organizations:

1. An organization is constituted of agents (individuals) that manifest a behavior.
2. The overall organization may be partitioned into partitions that may overlap (we will call these partition groups from now on)
3. Behaviors of agents are functionally related to the overall organization activity (concept of role).
4. Agents are engaged into dynamic relationship (also called patterns of activities [6]) which may be "typed" using a taxonomy of roles, tasks or protocols, thus describing a kind of supra-individuality.
5. Types of behaviors are related through relationships between roles, tasks and protocols.

An important element of organizations is the concept of *role*. A role is a description of an abstract behavior of agents. A role describes the constraints (obligations, requirements, skills) that an agent will have to satisfy to obtain a role, the benefits (abilities, authorization, profits) that an agent will receive in playing that role, and the responsibilities associated to that role. A role is also the placeholder for the description of patterns of interactions in which an agent playing that role will have to perform (in this paper, we do not distinguish between role and role assignment as in [12]).

Organization may be seen at two different levels: at the organizational (or social) level and at the concrete (or agent) level (from [3]):

We will call *organizational structure* [11] (or simply structure, if there is no ambiguity) what persists when components or individuals enter or leave an organization, i.e. the relationships that makes an aggregate of elements a whole. Thus, the organizational structure is what characterizes a class of concrete organizations at the abstract or organizational level.

Conversely, a *concrete organization* (or simply organization), which resides at the agent level, is one possible instantiation of an organizational structure. This is a realization consisting of entities that effectively take part in a whole, together with all the links that bring these agents into association at any given moment. It is possible to relate an organizational structure to a concrete organization, but the same organizational structure can act as a basis for the definition of several concrete organizations

An organization consists in two aspects: a *structural aspect* (also called static aspect) and a *dynamic aspect*:

Paper presented at AOSE'2003 (Agent Oriented Software Engineering), Melbourne July 2003, to be published in LNCS, Springer Verlag, 2003.

The structural aspect of an *organization* is made of two parts: a *partitioning structure* and a *role structure*. A partitioning structure indicates how agents are assembled into groups and how groups are related to each other. A role structure is defined, for each group, by a set of roles and their relationships. This structure defines also the set of constraints that agents should satisfy to play a specific role and the benefits resulting to that role. The *dynamic aspect* of an organization is related to the institutionalized patterns of interactions that are defined within roles. It defines also:

1. the modalities to create, kill, enter groups and play roles;
2. how these modalities are applied and how obligations and permissions are controlled;
3. how partitioning and role structures are related to agents' behaviors.

3.2 General principles of OCMAS

Previous sections have allowed us to understand the basic concepts of organizations. It is now time to consider multi-agent systems from an organizational perspective. The question now is: what are the main principles from which organization centered multi-agent systems (OCMAS) may be approached for both analysis and design?

The use of organizations provides a new way for describing the structures and the interactions that take place in MAS. The organizational level, the way organizations are described is thus situated in another level than the agent level that is often the only level considered in ACMAS. This level, which may be called "organizational level" (or "social level" as in [9]) is responsible for the description of the structural and dynamical aspects of organizations. This organizational level is an abstract representation of the concrete organization, i.e. a specification of the structural and dynamical aspects of a MAS, which describes the expected relationships and patterns of activity which should occur at the agent level and therefore the constraints and potentialities that constitute the horizon in which agents behave.

Principle 1: The organizational level describes the "what" and not the "how". The organizational level imposes a structure into the pattern of agents' activities, but does not describe how agents behave. In other terms, the organizational level does not contain any "code" which could be executed by agents, but provides specifications, using some kind of norms or laws, of the limits and expectations that are placed on the agents' behavior.

Principle 2: No agent description and therefore no mental issues at the organizational level. The organizational level should not say anything about the way agents would interpret this level. Thus, reactive agents as well as intentional agents may act in an organization. In other words, ant colonies are as much organizations as human enterprises. Moreover, seen from a certain distance, or using an intentional stance [2] it is impossible to say if the ants or the humans are intentional or reactive. Thus, the organizational level should get rid of any mental issues such as beliefs, desires, intentions, goals, etc. and provide only descriptions of *expected* behaviors.

Paper presented at AOSE'2003 (Agent Oriented Software Engineering), Melbourne July 2003, to be published in LNCS, Springer Verlag, 2003.

Principle 3: An organization provides a way for partitioning a system, each partition (or groups) constitutes a context of interaction for agents. Thus, a group is an organizational unit in which all members are able to interact freely. Agents belonging to a group may talk to one another, using the same language. Moreover, groups establish boundaries. Whereas the structure of a group A may be known by all agents belonging to A, it is hidden to all agents that do not belong to A. Thus, groups are opaque to each other and do not assume a general standardization of agent interaction and architecture.

These principles are not without consequences:

1. An organization may be seen as a kind of dynamic framework where agents are components. Entering a group/playing a role may be seen as a plug-in process where a component is integrated into a framework.
2. Designing systems at the organizational level may leave implementation issues, such as the choice of building the right agent to play a specific role, left opened.
3. It is possible to realize true "Open System" where agent's architecture is left unspecified.
4. It is possible to build secure systems using groups as "black boxes" because what happens in a group cannot be seen from agents that do not belong to that group. It is also possible to define security policies to keep undesirable out of a group.

4 AGR: a basic model of OCMAS

In order to show how these principles may be actualized in a computational model, we will present the basics and methodology of the Agent/Group/Role model, or AGR model for short, also known as the Aalaadin model [4] for historical reasons. We show that this model complies with the OCMAS general principles that we have proposed in the previous section.

4.1 Definitions

The AGR model is based on three primitive concepts, Agent, Group and Role that are structurally connected and cannot be defined by other primitives. They satisfy a set of axioms that unite these concepts.

Agent: an agent is an active, communicating entity playing *roles* within *groups*. An agent may hold multiple roles, and may be member of several groups. An important characteristic of the AGR model, in accordance with the principle□ above, is that no constraints are placed upon the architecture of an agent or about its mental capabilities. Thus, an agent may be as reactive as an ant, or as clever as a human.

Group: a group is a set of agents sharing some common characteristic. A group is used as a context for a pattern of activities, and is used for partitioning organizations. Following principle 3, two agents may communicate if and only if they belong to the same group, but an agent may belong to several groups. This feature will allow the definition of organizational structures.

Role: the role is the abstract representation of a functional position of an agent in a group. An agent must play a role in a group, but an agent may play several roles. Roles are local to groups, and a role must be requested by an agent. A role may be played by several agents.

4.2 Axioms

We note by x.send(y,m) the action of an agent x sending a message m to an agent y, by roleIn(r,g) the statement that the role is defined in a group g, and by plays(a,r,g) the statement that the agent a plays the role r in g. We also note by GStruct(g,gs), the statement that g is a group considered as an instance of the group structure gs, and member(x,g) the statement that an agent x is a member of a group g. Here are the axioms of the structural aspect of the AGR model:

a) Every agent is member of a (at least one) group:

□x:Agent, □g:Group, member(x,g)

b) Two agents may communicate only if they are members of the same group:

□x,y:Agent, □m:Message, x.send(y,m) □ □g:Group, member(x,g) □ member(y,g)

c) Every agent plays (at least one) role in a group:

□x:Agent, □g:Group □ □r:Role, plays(x,r,g)

d) An agent is a member of the group in which it plays a role:

□x:Agent, □g:Group, □r:Role
    plays(x,r,g) □ member(x, g)

e) A role is defined in a group structure:

□x:Agent, □g:Group, □r:Role, plays(x,r,g) □ □gs:GroupStructure □
    GStruct(g,GS) □ roleIn(r,GS)

Roles may be described as in Gaïa [15] by attributes such as its cardinality (how many agents may play that role). It is also possible to describe *structural constraints* between roles. A structural constraint describes a relationship between roles that are defined at the organizational level and are imposed to all agents. In AGR, we propose two structural constraints: *correspondence* and *dependence*. A correspondence constraint states that agents playing one role will automatically plays another role. For instance, to express the, quite classical, political correspondence between delegates of smaller groups (states, departments, regions) which are automatically members of another group where they act as representative (deputies, ambassador, etc.) we would use the following statement:

Role('delegate',GS1) □ Role('representative',GS2) where GS1 and GS2 are group structures. This constraint may be defined as follows:

□x:Agent, □g:Group, where GroupStructure(g,GS1), □g':Group where Group-Structure(g',GS2) such that: plays(x,'delegate',g) □ plays(a,'representative',g')

If the two roles have the same set of members, we will use the notation □ . For instance, in most human organizations (associations, corporation, syndicates, etc.), all voters are eligible. In our notation, we would express this constraint as:

role('voter',GS1) □ role('eligible',GS1)

whose definition is as follows:

□x:Agent, □g:Group, where GroupStructure(g,GS1), plays(x,'voter',g) □ plays(x,'eligible',g) □ plays(x,'eligible',g) □ plays(x,'voter',g)

Dependence constraints express dependencies between group membership and role-playing. For instance, an agent is authorized to be a director of a Laboratory only if it is also a researcher in the lab. This would be expressed in the following way:

Role('director','Lab') requires Role('researcher','Lab')

Its semantics could be defined in a 1st order logic as follows:

□x:Agent, □g:Group, where GroupStructure(g,'Lab'), plays(x,'director',g) □ plays(a,'researcher',g')

The AGR meta-model is represented figure 1 in UML.

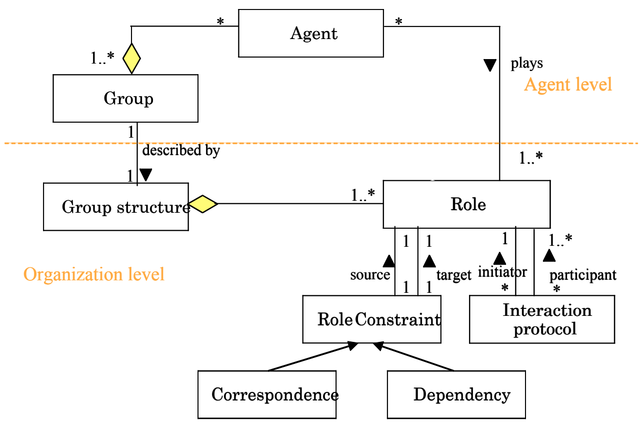

Fig. 1. The UML meta-model of AGR

Several notations may be used to represent organizations. In [13] a notation based on UML has been proposed to represent groups and roles. This is a very convenient notation to represent the abstract structures of an organization, but concrete organizations cannot be represented in this notation. This is why we will use the following another notation, that we call the *cheeseboard diagram*, which is very convenient to represent examples of concrete organizations.

Paper presented at AOSE'2003 (Agent Oriented Software Engineering), Melbourne July 2003, to be published in LNCS, Springer Verlag, 2003.

4.3 The "cheeseboard" diagram

In the cheeseboard diagram, a group is represented as an oval that looks like a board. Agents are represented as skittles that stands on the board and sometimes go through the board when they belong to several groups.

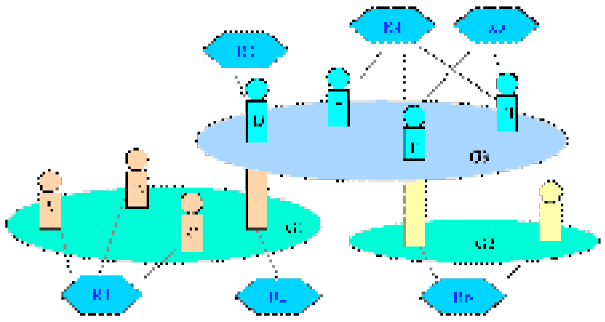

Fig. 2: The "cheeseboard" notation for describing concrete organizations

A role is represented as a hexagon and a line links this hexagon to agents. Figure 2 gives an example of a concrete organization using the cheeseboard diagram. In this picture, the agent F is a member of both G2 and G3, playing roles R4 and R5 in G2, and R6 in G3.

4.4 Describing organizational structures

The cheeseboard notation, while very adapted for concrete organization, is not suited to the description of relationships within organization at an abstract level, i.e. for the definition of organizational structures. Thus, we have introduced a notation for describing organizational structures.

In order to express organizational diagrams in a more simple and convenient way, we propose a set of graphical items. In this notation, *group structures*, i.e. abstract representation of groups, are represented as rectangles in which roles, represented as hexagons, are located. Constraints are represented as arrows between roles. We use two kinds of arrows. Large arrows are used for correspondence and thin arrows are use for modeling dependencies.

Interaction diagrams, which are represented as rounded rectangles, are used to describe communication protocols between roles. Without considering the way agents communicate, it is possible to describe communications at an abstract level, i.e. as specific constraints between roles. An interaction may takes place between two or more agents and is described at the organizational level between roles. The role initiator of the interaction is represented by an arrow that points towards the interaction. Other participating roles are represented as simple lines between interaction and roles. The figure 3 shows an organizational structure related to the concrete organization of figure 1. In this diagram, many different cases are represented. There are 3 group structures, called GS1, GS2 and GS3. The dependency d1 expresses a correspondence between the role R2 of GS1 and the role R3 of GS2. This allows for the definition of

Paper presented at AOSE’2003 (Agent Oriented Software Engineering), Melbourne July 2003, to be published in LNCS, Springer Verlag, 2003.

agents that act as representative between two groups. The dependency d2 expresses a dependency between R4 and R5, which means that all agents playing R5 must play R4. Interactions I2, I5 and I6, which are related to only one role, will be performed by different agents playing the same role. The interaction I3 takes place between agents playing three roles. Interactions may be figured by different types of diagrams: automata, Petri nets or sequence organizational diagrams, as we will see below.

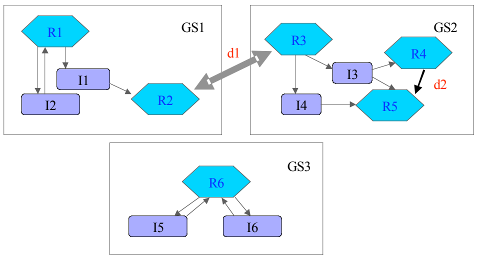

Fig. 3. Organizational structure representation

4.5 Describing organizational activities

To describe the dynamics of organizations, i.e. the temporal relation that is expressed between organizational events, such as the creation of groups, the entering or leaving of a group by an agent or the acquisition of a role in relation, we will use a specific notation, that we call *organizational sequence diagram*, which is a variant of the sequence diagram of UML (or AUML) [1].

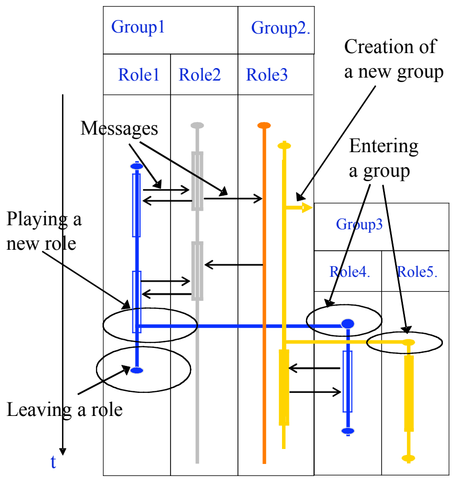

Fig. 4. The organizational sequence diagram

Paper presented at AOSE'2003 (Agent Oriented Software Engineering), Melbourne July 2003, to be published in LNCS, Springer Verlag, 2003.

Whereas in AUML vertical lines correspond to agents, in our diagram, the life of an agent is made of several segments of the same color (unfortunately, colors are displayed as gray levels in this paper). Each segment describes the "life" of an agent playing a specific role in a specific group. Thus, it is possible to represent the fact that an agent may belong to several groups and play several roles at once. Figure 4 shows a general view of this type of diagram.

4.6 Groups dynamics

Groups may be built at will. A group is created upon request of an agent, from an already described group structure. A group structure may be 'blank', thus allowing agents to build roles at will and to enter groups without any limits. However, in the general case, entering a group is a rather complex process, because an agent has to be authorized to enter a group. Due to axiom b) an agent cannot communicate directly to agents belonging to the group. Thus, it cannot request a permission to enter a group to agents belonging only to that group. A solution to this problem lies in the organization itself, in its possibility to build complex organizational structures. We will assume that an agent is permitted to enter a group only if it provides the right authorization. This agent could get this authorization in an "examination" like organizational pattern. An 'entrance' group, associated to the group A, acts as an "air lock" between the group A and its exterior. There is no authorization required for A to get the 'candidate' role in an entrance group. The 'gatekeeper' agent could then check the conformity of this agent to the specification of the structure and roles of the group A. Figure□ shows this adhesion process using a cheeseboard diagram. The semantics of this process has been described in [5] using a variant of the □-calculus.

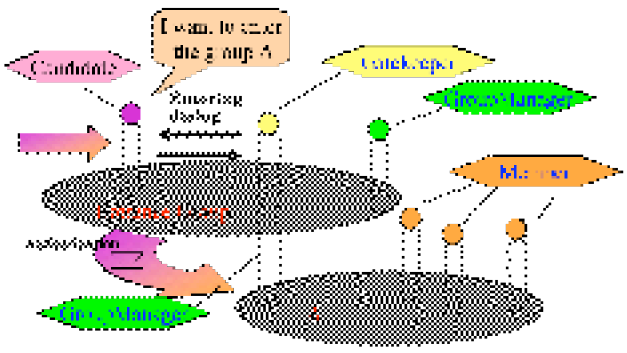

Figure 5. The cheeseboard representation of a group adhesion process.

It should be clear that this is only a simple aspect of all the organizational patterns that could be used to manage the organizational activities of an OCMAS. We just wanted to show that it is not necessary to relate to mental issues such as beliefs or goals to express the dynamics of an organization, and that *it is possible to manage an OCMAS using only OCMAS features*!

Paper presented at AOSE'2003 (Agent Oriented Software Engineering), Melbourne July 2003, to be published in LNCS, Springer Verlag, 2003.

Obviously, when it will come to implementation of agents, designers would have to relate the architecture and the cognitive properties of their agents to the organizational structure and dynamics of such a system. We only claim that, in OCMAS, this aspect would be considered in a second phase.

5 Methodology

Notations are not sufficient to describe a methodology. In this paper, we will only briefly suggest the key point of how a methodology could be defined on an OCMAS model.

The designer should first identify the main groups of the application. A group may be used for two main purposes:

- **To represent a set of similar agents**. In this case, a group is merely a collection of agents that exhibit certain similarities. There are usually few roles and a role may contain many agents. For instance, in AGR, to have a set of agents using the same communication language, such as ACL FIPA, one could design a FIPA group. Then the FIPA agents called the Directory Facilitator (DF) and the Message Transport Service (MTS) would be represented as agents playing the DF and MTS roles respectively. All other agents would merely have a simple 'member' role.
- **To represent a function based system**: each role then corresponds to a function or a subsystem of a whole system. Agents then act as specialists characterized by their skills to achieve functions associated to the roles. For instance in a computer network, printers have the ability to print and may be associated to the role of 'printer'. A soccer robot team would have the roles 'goalkeeper', 'leader', 'attacker', 'middle', etc.

Once these groups have been identified, the overall organizational structure is built using some organizational patterns [7, 11] such as the e-commerce organizational pattern that is presented in the next section as an example.

The partitioning of agents describes the way an organization is decomposed into its sub-components, and optionally the way these sub-components are further decomposed into their own sub-components, and the way these sub-components are aggregated. In AGR, hierarchies of groups, also called holarchy by Odell and Parunak [13] where a group is represented by an agent at the next level, may be represented by an organizational pattern where some 'delegate' agents in one group are seen as 'representative' agents in another group.

When the organizational structure is built together with organizational dynamic of group creation and adhesion, it is time to get into the definition of roles in a functional way. Then one could use the Gaia [15] methodology to fill the roles and relate them to the general structure. Our vision has some connection with object-oriented design, where the key diagrams are the class diagrams, which represent the static aspects of objects, and the sequence diagrams, which gives an insight of the dynamic aspects of objects. We use the same kind of distinction with the organizational structure diagrams and the organizational sequence diagrams. However, we often use the

Paper presented at AOSE'2003 (Agent Oriented Software Engineering), Melbourne July 2003, to be published in LNCS, Springer Verlag, 2003.

cheeseboard diagram to get a first idea of the organizational patterns one could use to build an OCMAS.

6 Examples

In order to explain how the AGR model may be used for analyzing and designing multi-agent systems, we will study two very-well known examples. The first one is a simplification of the well-known "travel agency" example that we have abstracted into a simplified but mode general e-commerce example. Clients try to get products from an agency considered as a brokering agent for product providers.

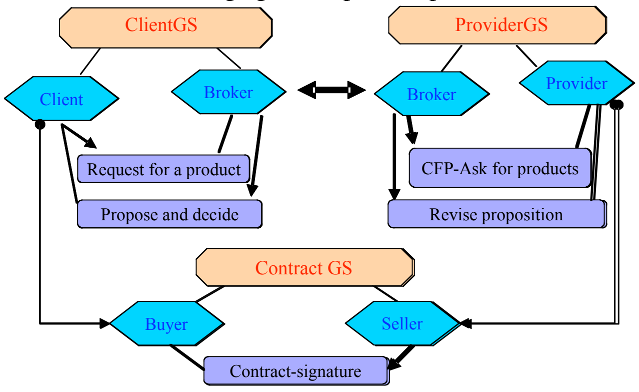

Figure 6. The organizational structure diagram of an e-commerce example.

We envision interaction between three group structures. The group structure of the clients, let us call it **ClientGS**, that interact with the broker; the group structure of the providers, let us call it **ProviderGS**, that interact with the broker, and the group structure of contracts, called **ContractGS**, which is used when a client decides to buy a product of the provider. The structure diagram is given figure 6.

When an agent enters a client group, the client talks to the broker and asks for a product. Then the broker (the same agent that plays the two 'broker' role in the client and the provider group) sends a call for proposal to providers. Then the proposals are presented to the client which decides which proposal to choose (this could lead to more interactions, asking the providers to revise their proposal). Then a contract group is created with both the client and the chosen provider, taking the respective role of 'Buyer' and 'Seller'. It is possible to represent this process, using the organizational sequence diagram (figure 7) which shows both the dynamic of the groups and the interaction protocol.

The second example is drawn from a situation that all researchers know very well: the "reviewing process" of papers in a conference. There are three group structures: the program committee group structure, the submission group structure (for a given conference there is only one group for each of these group structures), and the evaluation group structure. The program committee has only two roles: a program chair and a PC member. The submission group contains also two roles: submission receiver, which

Paper presented at AOSE'2003 (Agent Oriented Software Engineering), Melbourne July 2003, to be published in LNCS, Springer Verlag, 2003.

receives papers, and author. There are several submission groups, and the reviewing manager must be part of the program committee group. It is clear from this diagram that agents may belong to different groups: a committee member may be a reviewing manager of an evaluation group *and* an author submitting a paper.

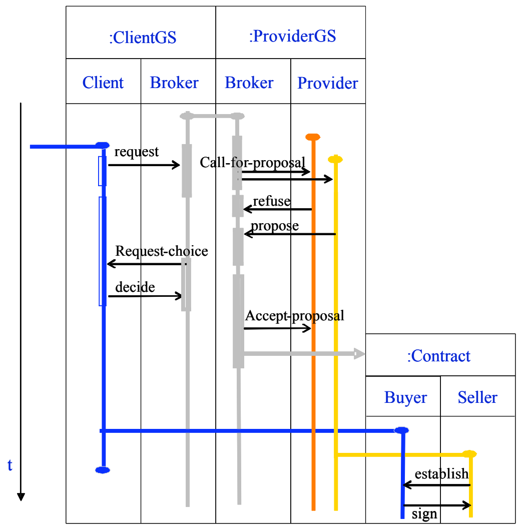

Fig. 7. Organizational sequence diagram of the e-commerce example

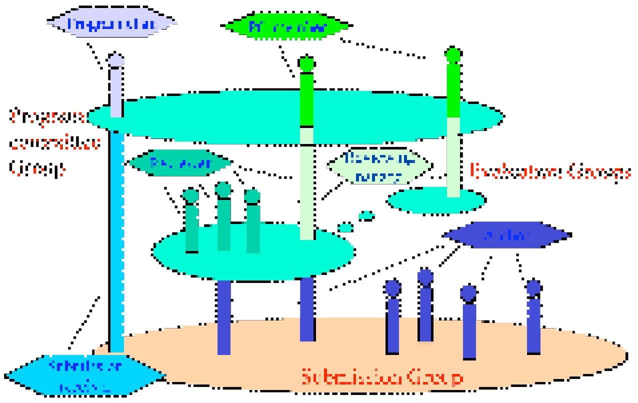

Fig. 8. The organization of a program committee using a cheese-board notation

The figure 8 presents the organizational structure of such an organization. Interactions such as 'distribute papers to review' or 'notification of acceptance' are protocols that relate agents through their roles. These protocols could be represented by any kind of diagram for representing protocols (finite state automata, Petri nets, etc.).

Figure 9, shows an organizational sequence diagram representing some part of the reviewing process. An author submits a paper to the submission receiver which is also the program chairman. As such, he/she asks a program committee member to review the paper. Then, this member creates a submission group and distributes the paper to the reviewers. When the reviewer has done its job, the committee member says that the paper is accepted (or rejected) and the submission receiver then sends an acceptance message to the author. Doing so, the author is therefore accepted as a speaker of the conference.

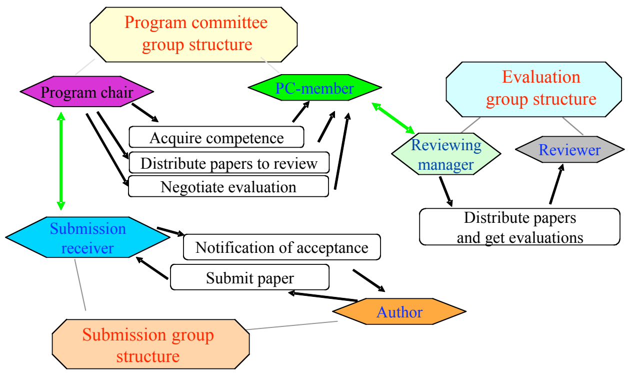
Program committee group structure
Program chair
PC-member
Evaluation group structure
Acquire competence
Distribute papers to review
Negotiate evaluation
Reviewing manager
Reviewer
Distribute papers and get evaluations
Submission receiver
Notification of acceptance
Submit paper
Author
Submission group structure

Fig. 8. The organizational structure diagram of the reviewing process example.

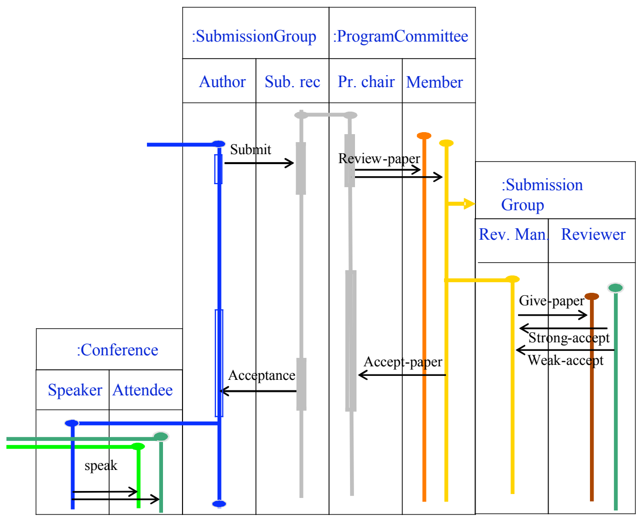

Fig. 9. A possible organizational sequence diagram of the reviewing process organization.

7 Conclusion

In the introduction, we have claimed that ACMAS have some drawbacks that OCMAS may resolve. For this reason, we have proposed a general framework to understand and design MAS based on organizational concepts such as groups, roles and interactions, which may overcome some of the weaknesses of ACMAS. We have presented the AGR model in this framework, showing how it is possible to design applications using these concepts that totally adhere to the OCMAS principles that we have introduced previously (see section 3.2):

1. The AGR architecture does not describe the "how", and only specifies the "what" by describing organizational structures made of group structures and roles.
2. We have not used mental issues such as intentions, goals or beliefs to describe the AGR model. We do not say that we should not use them: only that it is possible to build complex MAS architecture without using them. It is then the responsibility of the design process to describe agents able to live and interact in such architectures.
3. AGR provides a way for partitioning a system through the concept of group.

Thus, the main drawbacks of ACMAS disappear: it is possible to build secure applications at the group level, by designing gate keeper roles that prevent unauthorized agents to enter a group, or by describing norms (obligations, permissions, interdictions) that are related to groups and roles (this latter feature will be presented in a forthcoming paper). Complex programs may be built by using groups as dynamic frameworks that agents may create, enter and leave at will during their lifetime. In software engineering terms, agents may now be considered as some kind of dynamic "components" that live in dynamic frameworks.

Moreover, we claim that AGR is certainly one of the smallest possible organizational models. The structure or roles, is left open for the moment, but may be extended by integrating the most recent propositions on the nature of roles (see for instance [12]).

We have presented a set of diagrams (organizational structure, "cheeseboard" diagram, and organizational sequence diagrams) which may represent the different aspects of OCMAS. We have also sketched how these concepts may be used in a methodology based on organizational principles.

Organizational concepts may be used for practical implementations. The MadKit platform [8] that we have designed is built around the AGR model. Since its first release, hundreds of users (thousands of downloads) have been able to use these organizational concepts (presented in a less rigorous way than here) to build applications in various areas.

Many aspects of organizations, such as functional views, deontic aspects (concepts of norms and institutions) and the use of reflection to build complex MAS platform have been left over and will be presented in future papers.

References

1. Bauer, B., Müller, J.P. and Odell, J., Agent UML: A Formalism for Specifying Multiagent Interaction. in Agent-Oriented Software Engineering, (2001), Springer, 91-103.
2. Dennett, D.C. The Intentional Stance. M.I.T. Press, Cambridge, Massachusetts, 1987.

Paper presented at AOSE'2003 (Agent Oriented Software Engineering), Melbourne July 2003, to be published in LNCS, Springer Verlag, 2003.

3. Ferber, J. Multi-Agent Systems: an introduction to distributed artificial intelligence. Addison-Wesley, 1999.

4. Ferber, J. and Gutknecht, O., Aalaadin: a meta-model for the analysis and design of organizations in multi-agent systems. in Third International Conference on Multi-Agent Systems, (Paris, 1998), IEEE, 128-135.

5. Ferber, J. and Gutknecht, O., Operational Semantics of a Role-Based Agent Architecture. in Agent Theories, Architectures and Languages, (Orlando, 2000), Springer-Verlag.

6. Gasser, L. An Overview of DAI. in Gasser, L. and Avouris, N.M. eds. Distributed Artificial Intelligence: Theory and Praxis, Kluwer Academic Publishers, 1992, 9-30.

7. Giorgini, P., Kolp, M. and Mylopoulos, J., Organizational Patterns for Early Requirement Analysis. in IEEE Joint Int. Requirements Engineering Conference (RE'02), (Essen (Germany), 2002).

8. Gutknecht, O., Michel, F. and Ferber, J., Integrating Tools and Infrastructure for Generic Multi-Agent Systems. in Autonomous Agents 2001, (Boston, 2001), ACM Press, 441-448.

9. Jennings, N.R. On Agent-Based Software Engineering. Artificial Intelligence, 117 (2). 277-296.

10. Jennings, N.R. and Wooldridge, M. Agent-Oriented Software Engineering. in Bradshaw, J. ed. Handbook of Agent Technology, AAAI/MIT Press, 2000.

11. Mintzberg, H. The Structuring of Organizations. Prentice-Hall, 1979.

12. Odell, J. and Parunak, H.V.D., The Role of Roles in Designing Effective Agent Organizations. in Software Engineering for Large-Scale Multi-Agent Systems, (2003), Springer.

13. Parunak, H.V.D. and Odell, J., Representing Social Structure in UML. in Agent-Oriented Software Engineering II, (Montreal Canada, 2002), Springer, 1-16.

14. Rocha Costa, C. and Demazeau, Y., Toward a Formal Model of Multi-Agent Systems with Dynamic Organizations. in ICMAS'96, (Kyoto, 1996), AAAI Press.

15. Wooldridge, M., Jennings, N.R. and David, K. The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems, 3 (3). 285-312.

16. Zambonelli, F. and Parunak, H.V.D., From Design to Intentions: Signs of a Revolution. in AAMAS 2002, (Bologne (Italy), 2002), ACM Press, 455-456.