

A meta-model for the analysis and design of organizations in multi-agent systems

Jacques FERBER
ferber@lirmm.fr

Olivier GUTKNECHT
gutkneco@lirmm.fr

Laboratoire d'Informatique, Robotique et Micro-électronique
de Montpellier. Université Montpellier II, France

Abstract

This paper presents a generic meta-model of multi-agent systems based on organizational concepts such as groups, roles and structures. This model, called AALAADIN, defines a very simple description of coordination and negotiation schemes through multi-agent systems. Aalaadin is a meta-model of artificial organization by which one can build multi-agent systems with different forms of organizations such as market-like and hierarchical organizations.

We show that this meta-model allows for agent heterogeneity in languages, applications and architectures. We also introduce the concept of organizational reflection which uses the same conceptual model to describe system-level tasks such as remote communication and migration of agents. Finally, we briefly describe a platform, called MADKIT, based on this model. It relies on a minimal agent kernel with platform-level services implemented as agents, groups and roles.

1 Introduction

Whereas organization has been presented as a major issue of multi-agent systems, very few works have attempted to develop models of such systems using organizational approaches. Concepts such as roles, groups or organizations are used in a very informal, almost casual, way. Practically no effort has been made to incorporate organizational terms and concepts into multi-agent systems.

One of the rare attempts to include organizational concepts in multi-agent systems occurs in the work of L. Gasser. His early system, MACE, introduced the concept of role to describe multi-agent systems [9]. For instance, in the contract net protocol, agents with the role of manager make proposals to agents with the role of bidder. He also advocated the importance of computational organizational theories to fertilize multi-agent researches in an invited talk at ICMAS'95. Other examples of work on the use of or-

ganizational concepts in MAS come from M. Fox [7], and Lesser and Decker (see for instance [17]) but they do not tackle the problem of how to design (in software engineering terms) such organizations. Work on organizational theories has also been conducted in the context of the CommonKADS effort but has focused on human organizations models [2].

An exception is found in [1] which describes an analysis method for MAS in organizational terms, and in [10] which describes an algorithm for reorganizing societies of agents. A close, although more specific approach can also be found in [11].

This lack of interest in organizations comes from the general conceptual trend that defines multi-agent systems as mere aggregations of agents interacting together, where agent means an autonomous entity that behaves individually, almost selfishly.

Most of the work today in multi-agent systems has chosen to describe coordination and interaction structures from what could be called the "agent oriented" path. In that view, the designer of a multi-agent system is only concerned with agents' individual actions, and it is supposed that social structures come from patterns of actions that arise as a result of interactions. Moreover, it is often admitted that the behavior of an agent is a consequence of its mental state (beliefs, intentions, goals, commitments, etc.) (see for instance [16]). Even works with reactive agents [4] often take the agent-oriented view by considering that structures *emerge* from interaction between individual agents, even if these agents are not supposed to have mental states.

But, with the agent-oriented view it is often difficult to design complex systems because of the following problems:

- **Heterogeneity of language.** While heterogeneity has been considered as an important issue in multi-agent systems, there are no implemented systems, to our knowledge, that accept agents speaking different languages. Efforts to bring a standardized language such as KQML or ACL of the FIPA group, show that the problem of language heterogeneity has been consid-

ered so important that it required standardization methods to reduce language discrepancies.

- **Multiple applications and architectures.** No implemented systems, to our knowledge, allow several applications with different objectives and different internal architectures to work concurrently on the same platform.
- **Security.** While security is a major concern in distributed systems, no multi-agent techniques, to our knowledge, have been proposed to prevent agents from interacting with other agents. Any agent, in a system, is supposed to be able to interact with any other agent regardless of its capabilities, goals and authorizations.

In this paper, we claim on the contrary, that considering organizational concepts, such as groups, roles, structures, dependencies, etc. as first class citizens, and relating them to the behavior of agents is a key issue for building large scale and complex systems, and resolves all the previous problems in a very clear and efficient manner.

An organization can be defined, adapting [8], as a framework for activity and interaction through the definition of groups, roles and their relationships. Thus, we will regard an organization as a structural relationship between a collection of agents. In our view, an organization can be described solely on the basis of its structure, i.e. by the way groups and roles are arranged to form a whole, without being concerned with the way agents actually behave, and multi-agent systems will be analyzed from the “outside”, as a set of interaction modes.

Therefore, we will not address any issues about the architecture of agents, nor about the way agents act. Agents will only be defined by their functions in an organization, i.e. by their roles, and by the sets of constraints which they must accept to be able to play those roles. It does not mean that structures should be static, nor that groups should be totally separated. On the contrary, the power of structures comes from the ability of agents to join groups, and by doing so, to acquire new abilities that they would have not obtained otherwise. Their power lies also, as we will see below, in the fact that agents can play different roles in different groups, i.e. a computer scientist can also be a tennis player, an average musician and a good cook. Those activities are related to different communities in which he plays different roles and use different languages: the world of tennis does not use the same words and expressions as the world of computer science or the world of cooking, this becomes even more obvious when computer scientists speak English and cooks speak French.

AALAADIN is a generic meta-model of multi-agent systems based on the three main concepts of agents, groups and roles. It is a meta-model for describing organizations

because one can build different models of organizations in AALAADIN such as market-like or hierarchical organizations. Section 2 is devoted to the description of the meta-model (which we will call simply “model” for short. The reader should bear in mind that this model does not describe specific organizations, but a generic way to describe specific organizations).

As we will see in section 3, AALAADIN directly provides some sort of reflection that we call “organizational reflections”. This kind of reflection describes the operational aspects of groups, and system-level functions such as remote message passing, migration and resources management, through the use of meta-level groups and agents.

Section 4 briefly describes an implementation of AALAADIN in Java called MADKIT which allows for the description of several multi-agent systems working concurrently. System-level aspects, and the way agents, groups and roles are represented in the platform is described.

2 Conceptual Model

We assume in our model that the idea of collective structure permits two levels of analysis:

The concrete level corresponds to the core concepts of agent, group and role. It describes the actual agent organization.

The abstract, methodological level defines all possible roles, valid interactions, and structures of groups and organizations.

2.1 Core concepts

The AALAADIN model is based on three core concepts: *agent*, *group* and *role*. Figure 1 presents a diagram of this model.

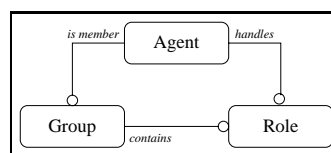


Figure 1. The core model

2.1.1 Agent

This model places no constraints on the internal architecture of agents and does not assume any formalism for individual agents. An *agent* is only specified as an active communicating entity which plays *roles* within *groups*.

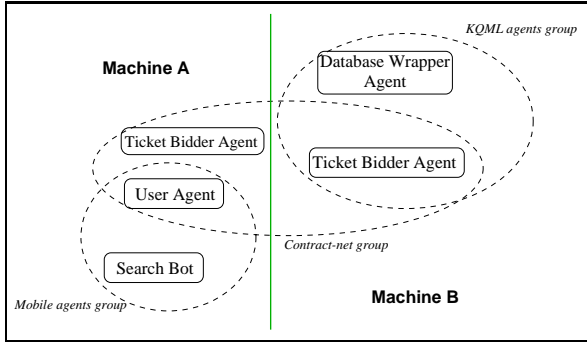


Figure 2. A scenario involving multiple groups

This agent definition is intentionally left general to allow agent designers to adopt the most accurate definition of agenthood relative to their application. The agent designer is responsible for choosing the most appropriate agent model.

2.1.2 Group

We define *groups* as atomic sets of agent aggregation. Each agent is part of one or more groups. In its most basic form, the group is only a way to tag a set of agents. In a more developed form, in conjunction with the role definition, it may represent any usual multi-agent system. An agent can be a member of n groups at the same time. A major point of AALAADIN groups is that they can freely overlap.

A group can be founded by any agent, and an agent must request its admission to an existing group. Groups might be distributed among several machines.

We will take as an example (figure 2) the classical application of travel assistance agents [20]. In our example, we will suppose that it is composed of agents involved in a specific contract-net protocol (to find the best offer for its user), with bidder agents (for travel assistance) communicating in KQML [6] with database agents. In our model, the “bidder” agent is part of both an application-dependant contract-net group as it is involved in this type of dialog, and a KQML group as it is able to communicate in this language. The user agent is part of a group gathering mobile agents, and simultaneously of the same contract-net group.

2.1.3 Role

The role is an abstract representation of an agent function, service or identification within a group. Each agent can handle several roles, and each role handled by an agent is local to a group. As with group admission, handling a role in a group must be requested by the candidate agent, and is not necessarily awarded. By relating communications to roles,

and by authorizing an agent to play several roles, our model allows agents to handle several heterogeneous dialog definitions simultaneously.

In the “travel assistance agent” example, cited in the last section, the *contract-net* group would contain two roles: the *bidder* role which is multiple and the unique *manager* role. This contract-net example also reveals that the communication model within this group can be easily described by identifying an abstracted interaction scheme between the “bidder” and the “manager” roles rather than between individual, actual agents.

A special role in a group is the *group manager* role, which is automatically awarded to the group creator. It has responsibility for handling requests for group admission or role requests. It can also revoke roles or group membership. By default, every request is fulfilled (i.e. if the agent does not specify specific group management behavior). It is the evaluator of the acceptance functions: an agent a can enter a group g to play a role r associated to a boolean acceptance evaluation function $f_{g,r}$, if and only if, $f_g(a)$ evaluates to *true*.

Notice that we do not define the particular mechanism for role access within a group. The following examples illustrate several possible functions controlling acceptance of a role within a group:

Systematic acceptance or refusal a *local* group gathering every agent running on an agent platform would always accept any agent

Acceptance conditioned by competences In this acceptance scheme, a role is awarded to an agent depending of on owned skills. Supposing that within a group each role r_i requires a set of competences $required(r_i) = \{c_1^i, c_2^i, \dots, c_n^i\}$, $f(a) \mapsto true$ if $\forall c \in required(r), c \in competences(a)$

Constrained by implementation where the candidate must exhibit some interface to be authorized to join the group.

Conditioned to an admission dialog where the request induces an interaction between the manager and the candidate to negotiate the admission.

Constrained by group status for instance by defining a coefficient of similarity between the current group members and the candidate (similar to trusted agents in [13]).

2.2 Methodological concepts

In addition to the core agent-group-role model, several concepts are added that are not represented directly in multi-agent organizations (figure 3). These additional concepts

only serve as an analysis and design tool. The purpose is to provide an organizational model specification from which an actual multi-agent system can ultimately be developed, and expressed with the core concepts.

In this section, we will detail the concepts of group and organizational structure, and their relation to the core model.

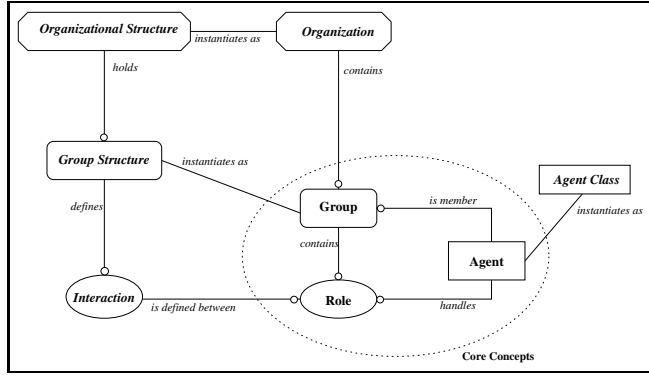


Figure 3. The methodological model

2.2.1 Group structure

The *group structure* is an abstract description of a group. It identifies all the roles and interactions that can appear within a group.

We define a *group structure* as a tuple: $S = \langle R, G, L \rangle$

- R is a finite set of roles identifiers. It represents the enumeration of all possible roles than can be played by agents within a group.
- G is the interaction graph. More precisely, it is a labelled oriented graph specifying the valid interactions between two roles. $G : R \times R \rightarrow \mathcal{L}$. The edge orientation corresponds to the role initiating the interaction. Each edge represents an interaction initiated by r_i with a role r_j and named *label*
- L is the interaction language. It is the chosen formalism for the individual interaction definitions. For each relation within the graph, we associate a unique protocol definition p to any edge $(r_i, r_j, label)$ within G . $\forall (r_i, r_j, label) \in G, \exists! p \in L$

We emphasize the fact that the group structure might be instantiated in a partial form in the actual group: all the roles defined in the group structure might not be present at a given moment in the group, depending on the group dynamics.

For instance, we can define a highly sophisticated market group with several broker, client and service roles, but the instantiated market group may only include a few of these roles.

2.2.2 Organizational structure

We define the *organizational structure* as the set of group structures expressing the design of a multi-agent organization scheme.

Thus, the organizational structure can be seen as the overall specification of the initial problem (i.e. conceiving an agent marketplace). Given the different group structures involved, the organizational structure permits the viable management of heterogeneity in agent communication languages, models and local applicative domains, within a single system.

We define an *organizational structure* as a couple $O = \langle S, Rep \rangle$ where:

- S is a set a valid group structures.
- Rep is the representative graph. It is a labelled graph where each edge $S_a S_b$ is labelled where two roles $r_1 r_2$, with $S_a \in S$ and $S_b \in S$, and where r_1 and r_2 are roles which are included in the set of roles defined in the group structures S_a and S_b , respectively.

Therefore, a representative structure definition between two groups A and B is an agent having simultaneously the role $R_{a,i}$ in group a and role $R_{b,j}$ in group b .

As with the relation between group structures and groups, we note that the actual organization is just one possible manifestation of the organizational structure. It might not include every group defined in the abstract organizational structure.

2.2.3 A market organization example

To clarify the inter-relations between structures and actual instances, and between organizational and group model, we present a short example of a market-like community modeled with these methodological guidelines (figure 4a).

We define three groups. The link between the two different interaction models is held by the broker role, which acts as a representative of the service provider group by the way of its broker role within this group. The clients interact with the broker to find a suitable service agent, and the contract group structure definition is aimed at ephemeral groups containing only the two agents involved in the final phase of a negotiation.

A possible state of a market organization resulting from this organizational structure is shown in figure 4b.

3 Reflection

There has been a large amount of work done on reflection in computer science. In [19], B. Smith coined the term “computational reflection” to describe a process that

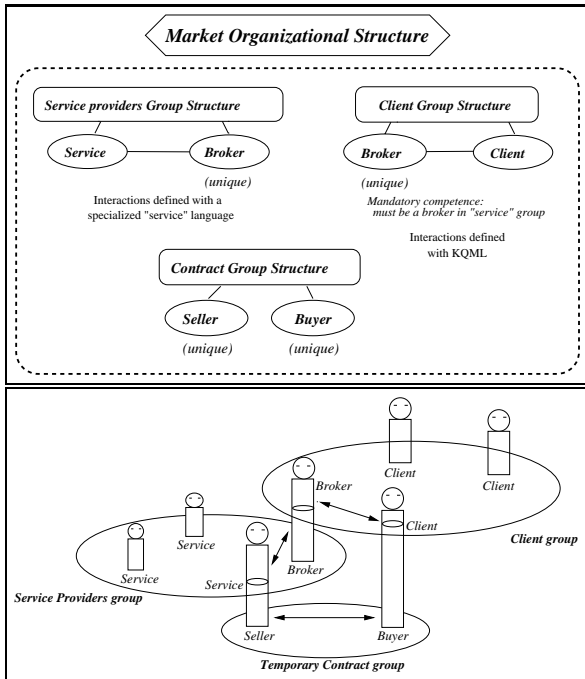


Figure 4. (a) The organizational and group structure of a market organization, (b) An actual market organization

is able to represent itself and reason about itself. Computational reflection has been applied to various languages and implementations [14]. Computational reflection has been shown to be very important representing system level activities such as resource management, task load balancing and object migration in parallel and distributed systems. An important body of work has been devoted to computational reflection in object oriented languages in general (see for instance [21]).

The most common implementation of procedural reflection for object oriented language is to use an *individual-based* model of reflection in which each object has its own meta-object which governs its computation [3]. An attempt to use this kind of reflection in the context of agent oriented languages can be found in [5]. But in this model it is difficult to represent the various aspects of an object (or an agent) with just one meta-object.

This limitation has led to the definition of the so-called *group-based model* of reflection in which the behavior of an object is realized at the meta-level by coordinated action of multiple meta-level objects [15].

But group-based reflection, while being very useful for representing and implementing various aspects of a set of distributed objects, requires a complete redesigning of the language and, in addition, is very resource consuming be-

cause of the great number of meta-level objects it requires.

In the following, we will introduce a new model of reflection, which we call *Organization-based model*, which overcomes all these difficulties.

This organization-based model is a direct consequence of our organizational meta-model. Thus, there is no need to introduce new concepts, such as "meta-level entities" or "meta-level groups", because all the reflective power comes directly from the fact that agents can belong to several groups at once and play different roles in different groups. All theories about reflection have to describe which entities and processes are reified, i.e. that are represented at the meta-level.

Reflection is performed by three general and complementary mechanisms:

1. **Cross-membership.** An agent can belong to both a domain-related group and a meta-level group which will handle system level activities such as management of resources, remote message passing, dynamic security, migration of agents, etc. Thus, meta-level operations are realized by entering a meta-level group. For instance, an agent will be able to migrate if it can enter the local group called *Mobility* and play the role *itinerant*.
2. **Agentification of services.** System level services are represented in AALAADIN as agents that play specific roles in meta-level groups. For instance, mobility is achieved through agents that play the migratory role in the local *Mobility* group.
3. **Use of representative.** An agent can be a representative of a group *A* in a group *B*. Thus, it is possible to transform an agent into a group if this is necessary and implement a kind of group-based reflection. All we have to do is to modify the behavior of an agent so that it delegates its tasks to agents of the group *B*.

As an example of the reflective behavior of AALAADIN, let us consider how the problem of migration is accomplished. Mobility is carried out by two groups. Not every agent within a system wants (or even can) be mobile. Expressing mobility as a specialized group allows for the description of mobility features in the same standard formalism of group and roles. Each site has a local group called a *Mobility-Group* which contains all the itinerant agents, i.e. agents that are potentially mobile, of a site *s* and a migratory agent which can make itinerant agents migrate. To play the *itinerant* role, and receive the *capacity* of being able to migrate, an agent needs to have the *competence* of being serializable. Thus, an agent that cannot be serialized cannot be accepted as an itinerant member of the group. There is only one agent with the migratory role in a *Mobility* group. This agent is responsible for

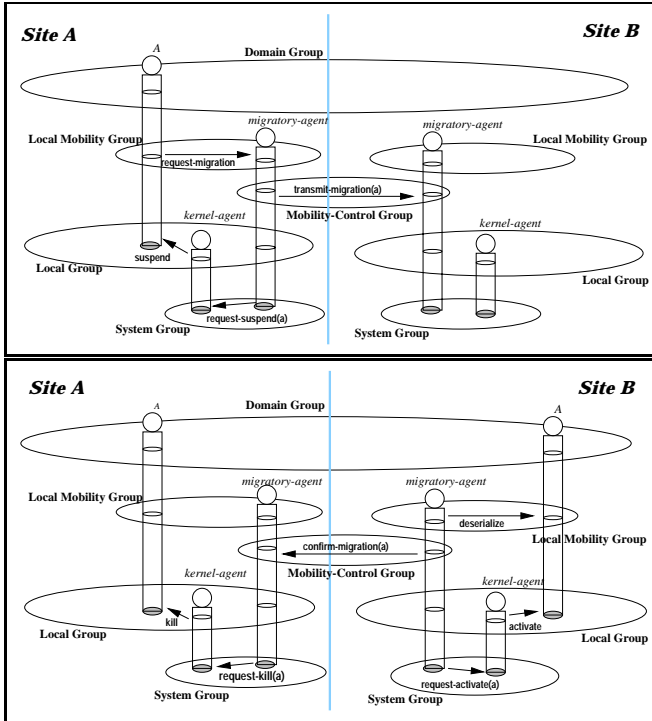


Figure 5. A migration scenario

the migration of other local agents on a remote platform. Migratory agents are also members of the *System* group, which contains agents that can control the resources allocated to other agents, to be able to control the life-cycle of other agents.

There is another group, called *Mobility-Control*, which gathers all agents with the migratory role in different local groups. This group eases all the communication and negotiation between migratory agents that are necessary to control the actual migration process.

Let us now consider a scenario of agent migration (figure 5). Let us suppose that there is an agent a that belongs to a domain group that resides in a site s . Let us suppose that for any reason a needs to migrate to another site s' (if there are too many agents on s , for instance).

Here is a typical scenario of agent migration:

- The agent a sends a request to the group manager of its local *Mobility-Group* to join the group with *itinerant* role. The group manager decides if the agent is suitable for migration, i.e. if it possesses the competence of being serializable.
- When this agent wants to migrate on another agent kernel, for any reason, a request is sent to the agent having the migratory role within the group to make a migrate. This request can be sent by a (if it can reason

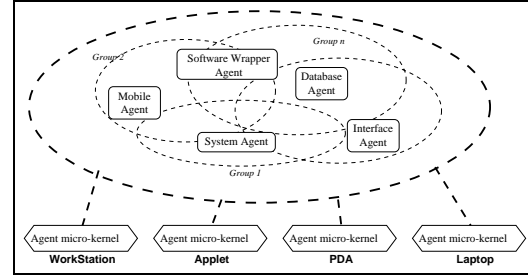


Figure 6. The MadKit platform Architecture

about such things) or by a system-level agent in charge of resource allocation.

- The migratory agent serializes the candidate, requests the kernel agent to suspend the activity of the candidate, and sends a to its peer on another site s' .
- The remote migratory agent deserializes the agent a , confirms good reception and asks the kernel agent to resume the new a . The local migratory agent which resides on s then asks the kernel agent to kill the old suspended agent.

4 The MadKit platform

To demonstrate the effectiveness of this model, we have built an agent platform [12], called MADKIT (for “Multi-Agent Development Kit”) to validate this approach. The MADKIT platform implements at its core, the concepts of agent, group and role, and adds three design principles:

- Micro-kernel architecture
- Agentification of services
- Graphic component model

The basic philosophy of the MADKIT architecture is to use wherever possible the platform for its own management: any services beside those assured by the micro-kernel are handled by agents, organized in groups, and identified by roles.

MADKIT runs on every platform supporting a Java 1.1 implementation.

4.1 Agent micro-kernel

The MADKIT micro-kernel is a small (less than 40 Kb) agent kernel. The term “micro-kernel” is intentionally used as a reference to the role of micro-kernels in the domain of OS engineering [18]. We would define an agent micro-kernel as a minimal set of facilities allowing deployment of agents services.

The MADKIT micro-kernel only handles the following tasks:

Control of local groups and roles The micro-kernel is responsible for maintaining correct information about group members and roles handled, and transmits admission requests to the appropriate group manager.

Agent life-cycle management The micro-kernel launches agents and has full control of their life-cycle. It also assigns the globally unique agent identifiers.

Local message passing Messages between agents are delivered via the micro-kernel facilities if the receiver is a local agent. If not, the message might be delegated to a specialized system agent.

The platform is not an agent platform in the classical sense. The reduced size of the micro-kernel, combined with the principle of modular services managed by agents, enable a range of multiple, scalable platforms (figure 6).

4.2 Agentification of services

4.2.1 Agents, groups and roles in the platform

Agents are defined by inheriting from an abstract class that provides agent identification, messaging API, and group and role related calls. These methods offer group creation, joining and various calls to identify which roles are present in a group, which agents are playing a given role, to make a request for role handling, delegation, or removal.

A few special groups and roles exist on the platform. The `local` group contains every agent running on the local micro-kernel. Admission to this group is automatic, thus finding local agents uses the standard group and role model and implementation.

The second special group, the `system` group, gathers members having the capacity to access the micro-kernel, thus potentially having control over life-cycles of other agents. Access to the system group is tightly restricted to agents identified by a trusted entity.

Interaction with the micro-kernel uses standard agent communication. The very first agent created at kernel bootstrap is a wrapper agent having full access and control on the agent micro-kernel. It founds and has the group-manager role for the `system` and the `local` groups. Then members of the `system` group can request privileged actions by interacting with it: group and role mechanisms are used to manage agent groups.

4.2.2 Agents services

In contrast to other architectures, MADKIT uses agents to achieve distributed message passing, migration control, dynamic security, and other aspects of system management.

This allows a very high level of customizing, as these service agents can be replaced without difficulty. For instance, it is possible to implement a completely different mechanism of distributed messaging without changing anything to the other agents on the platform. These services can also change at runtime by *delegating* roles to other agents.

The role delegation principle has the other interesting side-effect of allowing easy scaling. An agent can hold several roles at the beginning of a group, and as the group grows, launch other agents and delegate to them some of these roles.

4.2.3 Communication and distribution

Messaging, as well as group and role management, uses a unique agent identifier, and as this identifier is unique across distant kernels, MADKIT agents can be transparently distributed without changing the agent code. Groups can spread across different kernels, and agents usually do not even notice it.

Distribution in the MADKIT platform relies on agents handling two roles in the `system` group:

- The **communicator** is used by the micro-kernel to route non-local messages to other communicator agents, on distant platforms, which then inject the now-local messages into their kernel.
- The **group synchronizer** agents allow groups and roles to be distributed among kernels by distributing group and role changes to other synchronizers, which in turn enter this information into their local kernel. These group synchronizers use their own distributed group to ease distributed group management, after a bootstrap phase.

Since the distributed systems mechanisms are built as regular agents in the platform, communication and migration could be tailored to specific platform or application requirements only by changing the communication agents, for instance to manage disconnected modes for laptops. A MADKIT platform can run in full local mode by just not launching the communication agents.

These services are not necessarily handled by only one agent. For instance, the communicator agent can be the *representative* of a group gathering agents specialized in SMTP, sockets, or CORBA IIOP communications and delegate the task to the appropriate agent.

4.3 Graphic model

The MADKIT graphic model is based on independent graphic components. Each agent is solely responsible for its own graphical interface, in every respect (rendering, event

