The Knowledge Engineering Review, Vol. 25:4, 397–419. © Cambridge University Press, 2010
doi:10.1017/S026988891000024X

Role-based software agent interaction models: a survey

GIACOMO CABRI[1], LETIZIA LEONARDI[1], LUCA FERRARI[2] and FRANCO ZAMBONELLI[2]

[1]*Dipartimento di Ingegneria dell'Informazione, Università di Modena e Reggio Emilia, Via Vignolese 905 – 41125 Modena, Italy; e-mail: giacomo.cabri@unimore.it; letizia.leonardi@unimore.it;*
[2]*Dipartimento di Scienze e Metodi dell'Ingegneria, Università di Modena e Reggio Emilia, Via G. Amendola 2 – 42100 Reggio Emilia, Italy;*
e-mail: fluca1978@gmail.com; franco.zambonelli@unimore.it

Abstract

The role concept represents a useful approach that has been exploited in different agent-based systems, in particular applied to interactions between agents. There are some requirements that are important for the development of agent-based applications using roles, for instance the support for the analysis, the design and the implementation phases. We have considered and compared different role-based proposals in the literature, and this paper presents a survey of the most spread ones. We explain each proposal and point out if and how it meets the identified requirements. Far from deciding the best proposal, our aim is to present the advantages and drawbacks of several proposals to designers and developers, so that they can make the best choice with regard to their needs.

1 Introduction

Agents are autonomous entities able to perform their task without requiring a continuous user interaction (Luck *et al.*, 2003). Thanks to their *autonomy*, agents are exploited to build complex systems and applications where they are not restricted to play as involved entities, but can also play on behalf of their user(s). There are other features that, even if not essential for agents, can help them in their tasks, such as *mobility*, which enables them to move in a digital world made by hosts and networks, or *intelligence*, which allows them to learn and smartly react to the events in an execution environment (Wooldridge, 2002).

Even if, thanks to its autonomy, mobility and intelligence, an agent is a very powerful component, it does not execute alone in today's applications. In fact, a feature that is considered essential for agents is *sociality* (Wooldridge, 2002), which allows agents to interact with other entities in general, and with other agents in particular. Exploiting the sociality feature, today's applications rely on several agents, each interacting with the others in order to achieve the global goal. Therefore, such a kind of application is implemented by the so-called MASs (multi-agent systems), which are sets of agents designed to interact with each other, often in *open*, dynamic and *heterogeneous* environments. Coordination plays a fundamental role in MASs, since it allows agents to interact one with the other in a productive way. Since agents are autonomous entities, coordination is not a passive task, but actively involves agents themselves, which pass from 'coordinated entities' to 'coordinating entities'. In our work, *collaboration* is a particular kind of coordination.

Although coordination activities are very important tasks in MAS applications, embedding all the required logic in agents themselves seems to be an awkward solution, since it does not grant adaptability to environment changes and does not promote reusability. Furthermore, the choice of an embedded coordination schema does not meet today's agent-based application requirements, which include the capability to operate in open, dynamic and heterogeneous environments. To overcome the problems above, it is fundamental to use a coordination approach that is able to deal with application and developer needs, and that can be applied and reused in several scenarios.

There are several approaches related to agent coordination, including *tuple-spaces* (Cabri *et al.*, 2000), *group computation* (Hirsh *et al.*, 2003), *activity theory* (Omicini *et al.*, 2003) and *roles* (Cabri *et al.*, 2004b). In this paper, we focus on the use of the last, which can help developers to model the agent scenario in a manner similar to the real-life one. The main advantages of role-based approaches can be summarized as follows: First, they can be applied to existing entities to change their behavior; then, roles can be thought of as solutions common to different problems, so that they can be reused in different situations; further, roles enable a separation of concerns between the 'business' logic of the application, which is embedded in the agents, and the 'coordination' logic, which is embedded in the roles; finally, roles promote an organizational view of the system, which well suits agent-oriented approaches (Zambonelli *et al.*, 2001).

Roles have been already exploited in object-oriented approaches, where a role is defined as a *set of behaviors common to different entities* (Fowler, 1997), with the capability to apply them to an entity in order to change its capabilities and behavior. Other approaches promote roles as views of a particular object or entity (Baumer *et al.*, 1997), stressing the similarity between roles in computer programs and those in real life. There could be different concrete definitions of *role*, but, in abstract terms, it has the same meaning that common sense suggests and provides rights and duties (Biddle, 1979). Generally, a proposal defines a *model* of role and often its *implementation*, for instance, in terms of classes. Sometimes *descriptors* are provided, that is, entities that describe the roles and can be managed by the agents themselves. The term *role system* usually identifies an infrastructure or a platform that enables the assumption of roles by agents, either in a static or dynamic way.

Starting from previous works in object-oriented programming, roles have been applied to agents, which after all can be thought of as autonomous and active objects, promoting the reuse of solutions and making the definition of coordination scenarios easier. For instance, the *bidder*, *seller* and *auctioneer* roles can be reused in different auction applications, independently of the kind of auctions or of the agents' strategy. Roles allow the agent-application developers to model the execution environment so that agents actively 'feel' the environment itself. In other words, roles allow the developer and its agents to perceive in the same way the execution environment.

The importance of the use of roles is supported by the fact that they are adopted in different areas of the computing systems, in particular to obtain uncoupling at different levels. Some such areas are *security*, in which we can recall the Role-based access control (Sandhu *et al.*, 1996) that allows uncoupling between users and permissions, and the *computer supported cooperative work* (Tripathi *et al.*, 2002), where roles grant dynamism and separation of duties. In addition, in the area of software development, we can find approaches based on roles, especially in *object-oriented programming* (Kristensen & Østerbye, 1996; Demsky & Rinard, 2002), in *design patterns* (Fowler, 1997) and in computer-to-human interfaces (Shneiderman & Plaisant, 1994), which re-marks the advantages of role-based approaches.

When applied to the agent context, roles are mainly exploited to define common interactions between agents, for instance, the interactions between the contract-net participants. Roles embed all information and capabilities needed in a particular execution environment to coordinate, that is, communicate and collaborate with other entities and/or agents. Thanks to these features, an agent (and its developer) does not need to know details about the current execution environment but only which role to assume and use to interact with (or exploit) the environment itself. This leads to a separation of issues related to the agent logic and its coordination with other entities. The former is embedded in the agent itself, since it defines the agent base behavior, while the latter is embedded in a role and expresses an added behavior. This separation is more emphasized at the development phase, since it is possible to develop agents and roles they are going to use at separated times and ways, leading to a more modular development process. Another advantage of the use of roles is solution reusability. Since roles embed a behavior applied to a specific application context (e.g. collaboration in an MAS system), keeping it separated from agents, a set of roles can be successfully applied to other similar areas, leading not only to faster development, but also to a reuse of solutions.

Several role proposals for agents have been defined and implemented, each one with its own characteristics and applicability. The fact that several proposals exist can disorientate the developer when she aims at exploiting roles in her applications. In this paper, we survey and compare approaches for agent interaction based on the concept of role, considering the most known ones, emphasizing their advantages and drawbacks and outlining their usability in the development of agent applications. We consider only proposals where roles are the main focus, disregarding other approaches that, even if interesting, exploit roles as a means and do not provide models or tools to engineering them, such as Sierra *et al.* (2004) do, or that are too abstract and do not provide support in the development, such as Odell *et al.* (2003) do, or that are too simple (for instance, providing for bare role classes for the implementation), such as Cabri *et al.* (2003b) do, or finally those that are too bound to specific application fields. This paper presents an evaluation of the considered proposals, without the aim of scoring them as 'bad' or 'good'. In fact, this paper presents an analysis of the different proposals to help developers understand *when* and *how* they can be successfully used.

This paper is organized as follows: Section 2 details how the role proposals have been evaluated; Section 3 presents each proposal; and Section 4 reports the comparison of the considered proposals. Finally, Section 5 shows the conclusions of this survey.

2 Evaluation criteria

Since each role proposal has its own characteristics, in order to produce a better comparison among them, we will focus in particular on the following features:

1. Support for the application development, with particular regard to the main involved phases:
   a. *Analysis*: this phase outlines the system requirements, disregarding the implementation of specific issues.
   b. *Design*: during this phase, all involved roles are defined; that means that it is established which and how many roles will be used, which behavior everyone will provide, how they will be assumed and released and which relationships tie them.
   c. *Implementation*: this phase makes concrete the previous two, taking into account the exploitation of roles in the application.
2. Notation: the availability of a notation to describe roles and their behaviors can help the designer and developers to deal with roles, better understand and apply them. Furthermore, the use of a formal notation allows a deeper study of the system, even with automatic tools able to discover and define dependencies and collaboration requirements.
3. Dynamic role allocation: today's applications require a strong dynamism, and this is true also for agent applications, which require dynamic adaptability to execution environment changes. Roles can grant this adaptability, providing a way to interact and collaborate with the execution mediating the communication. Nevertheless, to achieve the required adaptability, roles should be used in a dynamic way, without requiring, for example, the definition of static code embedded in agents.
4. Openness: to be widely applicable, a role proposal must be open, which means that it should be possible to successfully use it also in environments and scenarios where not all the agents are well known at the design time.
5. Interoperability: means the capability to exploit roles in different agent platforms.

We have chosen these criteria on the basis of our experience in building agent-based applications and in analyzing role-based approaches.

3 Overview of surveyed proposals

Each of the following subsections describes a single role proposal we have evaluated, emphasizing the evaluation criteria each one supports (i.e. for which it provides appropriate models or tools to help the designer and developers in their work. We have decided to present the surveyed proposals

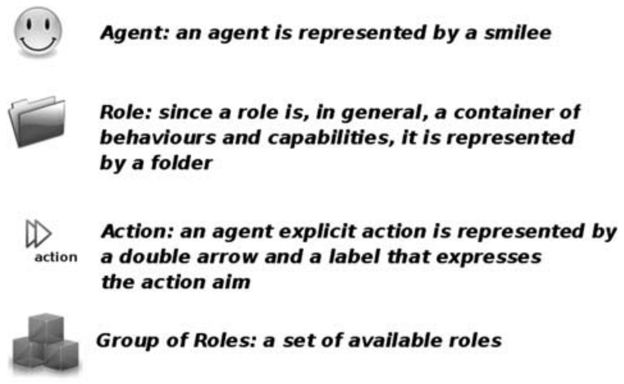
Agent: an agent is represented by a smilee
Role: since a role is, in general, a container of behaviours and capabilities, it is represented by a folder
action
Action: an agent explicit action is represented by a double arrow and a label that expresses the action aim
Group of Roles: a set of available roles

Figure 1 Symbols used in the proposal' figures

in alphabetic order, because (i) it is the most neutral one and does not introduce any specific bias, and (ii) the proposals' features are quite heterogeneous and any organization of the presentation could give more importance to some of them to the detriment of others.

In Figure 1, we report on the symbols exploited in the figures shown in the proposals' subsections.

3.1 AALAADIN

AALAADIN is a metamodel (Ferber & Gutknecht, 1998) to define models of organizations for agents (Ferber *et al.*, 2004), and thus works at a high level of abstraction. This model defines several main concepts:

- *Agents* are active communicating entities;
- *Groups* are sets of aggregated agents, which can be considered as an atomic entity. Agents can dynamically join, leave or create groups;
- *Roles* are abstract representations of an agent *service*, function or *identification* within a group.

In the AALAADIN model, the definition of agents is intentionally not detailed. In fact, leaving the agent definition free, agent designers and developers can use the one that better meets their requirements, supporting openness as well (criterion 4). Agents can assume multiple roles at the same time, but it is important to note that, since AALAADIN supports the concept of locality, each role is local to the group the agent belongs to. Figure 2 shows the AALAADIN metamodel emphasizing relationships among the above parts.

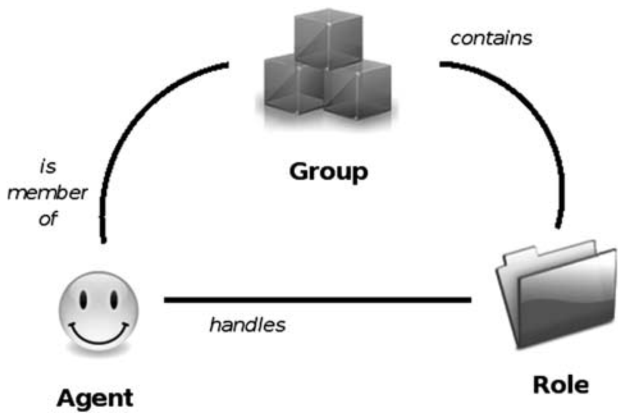

Figure 2 The AALAADIN metamodel

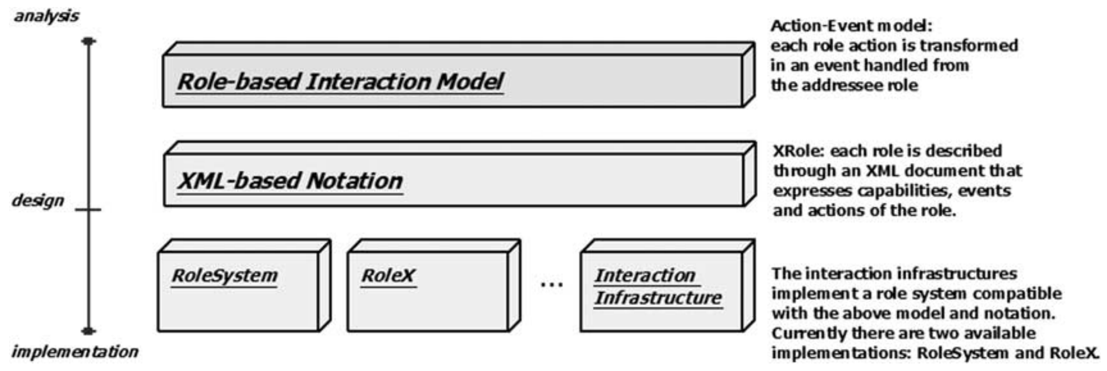
analysis
design
implementation
Role-based Interaction Model
XML-based Notation
RoleSystem
RoleX
...
Interaction Infrastructure
Action-Event model: each role action is transformed in an event handled from the addressee role
XRole: each role is described through an XML document that expresses capabilities, events and actions of the role.
The interaction infrastructures implement a role system compatible with the above model and notation. Currently there are two available implementations: RoleSystem and RoleX.

Figure 3 The behavioral role for agent interactions model layers

Since this proposal defines a metamodel, it can be used both at the analysis phase (criterion 1.1), helping analysts to understand the organization of the system, and at the design phase (criterion 1.2), supporting developers to recognize and model interactions involved in the application. Moreover, since this proposal relies on roles, there is separation of concerns between agents and their behaviors.

The advantages of AALAADIN cover three issues. First, heterogeneity of agent interactions is allowed by the definition of groups and related roles that enforce locality. Second, organizational models based on the AALAADIN metamodel can be reused for different applications with similar tasks or sub-tasks; moreover, applications with different architectures but compliant with the metamodel are allowed to work on the same platform. Finally, AALAADIN also addresses security, because constraints on the interactions between agents can be contemplated since the design phase of the development process.

Thanks to its metamodel, AALAADIN provides interoperability (criterion 5). Since roles are abstract representations of agent services, developers are free to use whatever implementation model they want, still keeping the implementation phase coherent with the AALADIN design.

A concrete adoption of the AALAADIN proposal can be found in the MadKit platform[1] from the same authors (Gutknecht & Ferber, 2000). This platform is based on a micro-kernel that provides the essential services for agents running in the platform, while other services are provided directly by the agents (this leads to an approach called 'agentification of services'). It is important to note that the AALAADIN proposal is used in MadKit only in the analysis and design phases and that there is no direct implementation of the metamodel in the platform (criterion 1.3 partially).

3.2 Behavioral role for agent interactions

The behavioral role for agent interactions (BRAIN; Cabri *et al.*, 2003a) is a proposal that defines a role as a set of *capabilities* and *expected behavior*. The former defines what an agent can actively do, while the latter defines what an agent must do according to the role it is playing.

BRAIN is an approach that covers all the development phases (criterion 1) explicitly defining a three-layer model (as shown in Figure 3), where each layer addresses a different development phase. The top layer defines a role-based interaction model, where agents' interactions and behaviors are embedded in roles. A role is defined as a capability enabler that exposes to the agent that plays it a set of *actions*, which allow the agent to interact with other agents/roles.

The BRAIN middle layer defines an XML notation to define roles and their capabilities (i.e. which actions and events are available and must be managed). Thanks to such a notation, called XRole, BRAIN matches criterion 2. It is important to note that the XRole notation defines all the semantic information about a role, which is what the role does, how it can be used to achieve a

[1] `http://sourceforge.net/projects/madkit`

specific aim, but not how it will do. In fact, the latter is the aim of the below level, interaction infrastructure, to implement the mechanisms behind the role actions and events.

Since the XRole notation represents role semantic information, developers can analyze and choose the most appropriate role for a specific task among those available. The choice of an XML notation is also important because it allows not only developers but also agents and automated tools to inspect XRole documents at runtime (criterion 3).

At the implementation level, BRAIN merges the above levels providing an implementation of the *action-event* model, where interactions happen by means of events, which are consequences of role actions, delivered to other roles. Addressee roles must manage the incoming events as their expected behavior. During an interaction, agents exploit their capabilities to perform some actions. These are translated by the interaction system into events, delivered to the addressee agent/role, which manages the incoming event showing an expected behavior.

Currently, BRAIN presents two implementations of the bottom layer (interaction infrastructures). Both of them define roles as first-class entities, but while the older one is based on a traditional object-oriented approach, the most recent one is based on runtime agent alteration, which later recalls AOP (aspect-oriented programming) (Kiczales *et al.*, 1997), even if everything happens at runtime without any compiler or meta-language definition and the entire process is customized for the agent scenario. In particular, the latter implementation introduces a new feature not provided by most other proposals, which is *external visibility*. Thanks to this, whatever agent can be immediately recognized as playing a given role (criterion 4), without doubt or need to ask the environment for such information.

Thanks to its multi-layer structure, BRAIN supports interoperability (criterion 5). This is emphasized, for example, by the capability to exploit several interaction infrastructures beneath the same model and notation levels. Interoperability is achieved also by means of the semantic value of XRole documents, which allows agents to select the best role depending on the task to be completed, and uncoupling therefore the role implementation from the role semantic. This also means that it is possible to model a role by mapping it onto different implementations. Even role maintenance is simplified, since agents will not directly see different role implementations and versions, accessing them by means of semantic information.

3.3 Fasli's proposal

Fasli's proposal (Fasli, 2003) joins several concepts, such as commitments and obligations (Castelfranchi, 1995), with the power of roles, promoting the use of a formal notation (criterion 2) and analysis of the applications (criterion 1.1).

The basic idea of this proposal is that multi-agent systems are composed of *social agents*, where 'social' means that they do not act in isolation. Please note that the term *social agent* refers to either a single agent or a group of correlated agents. In fact, social agents' decisions and acts can affect those of other agents, even if not intentionally, and thus this proposal presents a formal definition of the agent structure using roles and relationships among them. In these scenarios, commitments are fundamental concepts, since they express the wills of social agents.

When a single agent decides to join a 'social agent' (i.e. a group), it assumes a specific role, which is composed of (see Figure 4):

- *a set of social commitments* used to express the objectives within the social agent;
- *a set of obligations* that define what an agent is obligated to do either *to* other agents or *with* other agents. If an obligation is not honored, the counterpart agent can reserve the right to impose sanctions;
- *a set of rights*, similar to the set of obligations, but defines what an agent expects to receive from other agents.

The assumed role defines the 'social position' of the joining agent in the social agent, so that each agent in the social agent knows its position and plays according to it. Furthermore, this

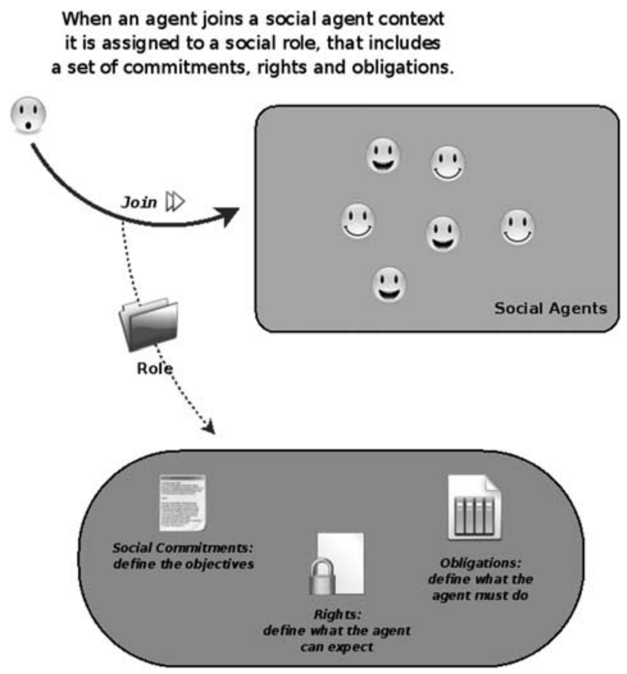
When an agent joins a social agent context it is assigned to a social role, that includes a set of commitments, rights and obligations.
Join
Social Agents
Role
Social Commitments: define the objectives
Obligations: define what the agent must do
Rights: define what the agent can expect

Figure 4 Fasli's role proposal

proposal does not exclude the possibility that an agent joins several social agents, thus playing multiple roles at the same time.

A social agent structure is formally defined as a tuple that contains a set of roles that can be played within the social agent, and a relationship graph that shows all valid relationships among the above roles. This definition of social agents is flexible, describing only roles and relationships among them; furthermore, the exact form of social agents depends on how agents interact with each other.

Commitments are considered to be the attitudes that hold social agents together, and their type depends on the particular instantiation of the social agent or on the conditions of its creation.

This proposal offers a comprehensive view of social and collective activities, describing them as social and collective concepts. Moreover, several concepts such as commitments, obligations and roles have been related to each other within a formal notation that helps designers. Nevertheless, there is no concrete support at the implementation level.

3.4 Gaia

The main aim of Gaia (Zambonelli *et al*., 2003) is to model multi-agent systems as *organizations* where different roles interact. Figure 5 shows the model exploited by Gaia in the application development. In Gaia, roles are considered only in the analysis phase (criterion 1.1).

Like other proposals, Gaia defines roles by means of associated attributes:

- *Responsibilities*, which specify the functionalities of agents playing such roles. Responsibilities are further divided into *liveness* properties (something good happens) and *safety properties* (nothing bad happens);
- *Permissions*, which are a set of rights associated with the role and, further, to the agent playing it;
- *Activities*, which are computations internal to the agents, which can be executed without interacting with other agents.
- *Protocols*, which specify how an agent playing such a role can interact with agents playing other roles.

Even if Gaia does not have an explicit notation to describe roles, the formal notation used to define permissions related to roles can be successfully exploited even at a higher level (criterion 2).

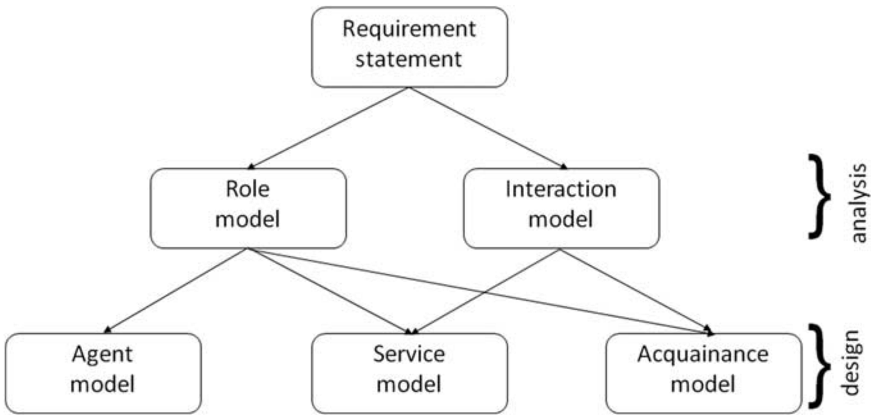

Figure 5 The Gaia models

In fact, this notation enables designers to define what a role can or cannot do, and can be further exploited to express liveness properties.

Gaia focuses on role interactions and supports a model to describe dependencies and relationships between different roles, based on the FUSION method (criterion 2). The above model is supported at the design phase (criterion 1.2 partially) and is made of a set of *protocol definitions*, each of them related to the kind of interaction among roles.

Gaia allows an organizational view of the system, thanks to the use of roles, enabling at the same time a separation between the application design and its real implementation. Nevertheless, in this proposal, agents involved in an interaction, and the protocols they are going to use, must be known *a priori*, ruling out dynamism and flexibility. Furthermore, even if roles are associated with permissions, there are no social rules that drive interactions.

3.5 Kendall's proposal

Kendall has done a lot of work about roles and agents (Kendall, 2000), and her proposal is perhaps the most complete. In her proposal, a role includes *capabilities* and *responsibilities*. Her effort aims at covering different phases of the agent-based application development, from the analysis to the implementation (criterion 1). To achieve this, her proposal exploits both object-oriented programming and aspect-oriented programming (AOP) (Kiczales *et al.*, 1997) and in particular their flexibility, reusability and dynamism (criterion 3) in order to grant these features also in the use of roles.

Starting from the analysis phase, this proposal exploits *role model catalogs*, which are collections of role models used in agent applications. Thanks to these catalogs, developers performing application analysis can search for and use the best solution for their application, choosing among several already implemented and tested solutions for similar scenarios. Nevertheless, to make this possible, as the author emphasizes, there is the need for appropriate role documentation, in order to facilitate the selection of roles.

Another important concept to keep in mind during the analysis phase is what the author calls the *roleification process*, which is the process of associating a role with a concept. This process can be compared with the classification of the object-oriented paradigm.

In the Kendall proposal, developers could, at the design phase, design agents by composing or assembling all features provided by single roles, obtaining a compound and complex entity. This phase is not explained in detail by the Kendall proposal.

At the implementation phase, the author proposes two separate approaches. The first is based on the role object pattern (Baumer *et al.*, 1997), which promotes the design and development of roles by traditional object-oriented classes and objects. The second is based on the exploitation of

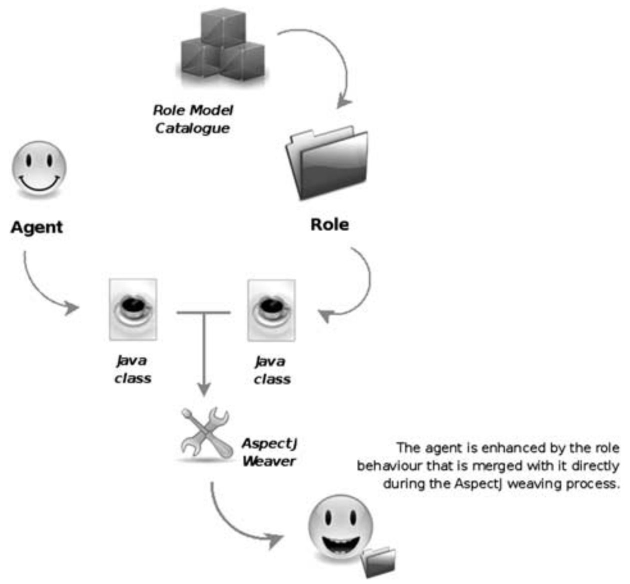
Role Model Catalogue
Agent
Role
Java class
Java class
AspectJ Weaver
The agent is enhanced by the role behaviour that is merged with it directly during the AspectJ weaving process.

Figure 6 How aspect-oriented programming can be used to implement agent roles

AOP, where the capabilities of this paradigm, which has not been explicitly designed in connection with roles, are exploited to dynamically manage agents and their roles (Communications of the ACM, 2001). *Aspects*, which are the core of the AOP, are entities orthogonal to those defined in the program, and that help the definition of crosscutting concerns. Aspects are bound with the standard classes by means of an *aspect weaver*, which usually works at the compile time, but can also work at class loading time (even with some limitations with regard to the compile-time execution). Since aspects change the behavior of the objects they are applied (joined) to, they can be considered as roles, and thus applying appropriate aspects (roles) to agents at runtime can provide a role-based behavior (see Figure 6).

Even if interesting for the technologies and concepts exploited, the Kendall proposal has some limitations. First of all, the use of AOP as possible implementation of roles leads to the need to learn a new paradigm. Furthermore, the definition of a role in AOP leads to possible not-reusability situations: in fact, the role/aspect to be joined to an agent must know the agent definition, and this implies that the role cannot be applied to different agents without modifications. Last but not least, AOP implementations require developers to know a new meta-language in order to define aspects (and thus roles), which usually differs from the development language and therefore requires a learning curve.

3.6 Role/interaction/communicative action

The role/interaction/communicative action (RICA) theory (Serrano & Ossowski, 2004) was born with the main aim of improving the Foundation for Intelligent Physical Agents (FIPA) standard[2] with support for social concepts. In fact, so far, FIPA specifications provide for interoperability (criterion 5) and open standards (criterion 4) of agent systems and communication techniques (such as Agent Communication Language—ACL), but do not provide any mechanism to model interactions from a social point of view.

Starting from the above consideration, the RICA theory joins communication and social concepts, merging FIPA-ACLs and organizational models, and keeping them as first class entities. RICA recognizes *communicative entities*, which are those that can be aggregated and organized in

[2] `http://www.fipa.org`

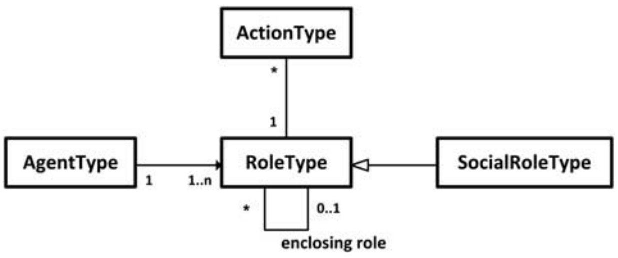

Figure 7 A part of the role/interaction/communicative action metamodel UML class diagram

a social way; in other words, communicative entities are agents that act in a society. Given the concept of communicative entities, the RICA theory can be split into two main components:

- a *metamodel*, which can be used as a language for specifying communicative entity types;
- a set of *structure and behavior* of each communicative entity.

While the metamodel is already defined in the RICA theory and in its implementation (RICA-J), the structure and behavior are not defined and need to be provided by the agent developer, which must thus decide exactly how her communicative agents must cooperate within the environment. In the RICA theory, the behavior of an agent is performed through its role(s) (Serrano & Ossowski, 2004); thus, an agent can be considered as a set of roles it will play during its life, and a role is conceived as a *social behavior*.

Figure 7 shows a part of the RICA metamodel class diagram, which emphasizes the main concepts of this proposal. First, as already written, each agent (type) is defined through the roles (types) it will play. Second, each role can perform one or more actions, implementing a social behavior. Actions can be specialized in *social* and *communicative* ones (not shown in Figure 7). Finally, each role can be specialized as a *social role*, which represents the behavior of agents interacting in a social context.

It is worth noting that the RICA theory makes a large use of 'social specializations', meaning that, for each concept, it is usual to find a 'social' corresponding one (e.g. *role* can be specialized in *social role*, *action* in *social action*, and so on). While social specializations can be very useful when modeling an environment from a social point of view, perhaps the 'communicative specialization', a specialization of a social entity, is the most important one. Communicative entities, with particular regard to communicative actions, are those used to perform such kind of interaction with the precise aim of being observed by an agent playing a specific role (Sadek, 1991). This realizes a complete interaction between agents, where the communicative agent communicates with the listener (the one playing a specific role).

The RICA metamodel can be used as a formal language (criterion 2), providing support for the analysis (criterion 1.1) and design (criterion 1.2) phases of agent applications. Thanks to the RICA-J (RICA Jade) implementation (criterion 1.3), this proposal is complete and can be used during all phases of the application development. RICA-J is in charge of providing concrete implementations for all RICA theory classes, and a kind of 'glue code' to run RICA-J over agent systems based on the Jade platform[3]. It also enables the dynamic management of roles (criterion 3).

The most relevant advantages of RICA and RICA-J are: (i) it is a complete theory and model that can be used during the whole development and (ii) it aims at overcoming some limitations of the FIPA specifications, and its implementation is based on a FIPA system, improving the adoption of such a model.

A drawback of the RICA approach is that it defines an agent mainly by means of the role it will play during its life; such a definition could clash with other agent models and theories forcing the

[3] http://jade.tilab.com

agent developer to a mental shift toward the new definition of an agent. Moreover, RICA-J exploits the FIPA directory facilitator (DF) in order to keep track of all assumed roles by each agent. Even if the DF could not become a bottleneck, the use of a centralized repository for 'the role status' of each agent tends to constrain interactions, since each agent, before starting an interaction, must contact the DF to know which agent can 'understand' the interaction, and it is not free to dynamically engage interactions with other agents. This also leads to less autonomous scenarios, since agents are playing roles, but the knowledge about the role played is decentralized with respect to the running agents.

3.7 Role-based evolutionary programming

The role-based evolutionary programming (RoleEP) treats cooperative mobile agents, which belong to the same application and collaborate to achieve a common goal (Ubayashi & Tamai, 2000). RoleEP defines an application as a set of collaborating agents, which cooperate together, through roles.

The main concepts of the RoleEP model are as follows:

- *Environment*: the environment represents the context where agents live and act, interacting with each other. The environment provides several services as meaning of attributes and methods, and make them available to agents through local roles;
- *Objects*: objects in RoleEP though can be according to the object-oriented paradigm, as for instance of classes with attributes and methods;
- *Agents*: an agent is a particular kind of object with one or more roles attached;
- *Roles*: roles are a part of the environment and they are made of *attributes*, *methods* and *binding interfaces*. Role attributes and methods can be whatever is needed to complete the agent task(s).

Starting from the above concepts, Figure 8 shows the RoleEP model. As it is possible to note from the figure, each object can be bound to one or more roles (in the figure, the bindings are shown with a bold arrow), and interactions among agents happen through such roles.

To achieve a good degree of flexibility and dynamism at runtime, RoleEP exploits *binding-interfaces*: such interfaces can be thought of as sets of abstract methods in charge of invoking the concrete methods on the object they are bound to. In this way, RoleEP allows agents to dynamically assume roles at runtime without worrying too much about the concrete implementation of the objects they are going to interact with, since they are only in charge of exploiting binding interfaces (criterion 3).

Even if the authors do not explicitly address the design phase of the application development, the RoleEP constructs seem to be useful to model systems where components (agents in particular)

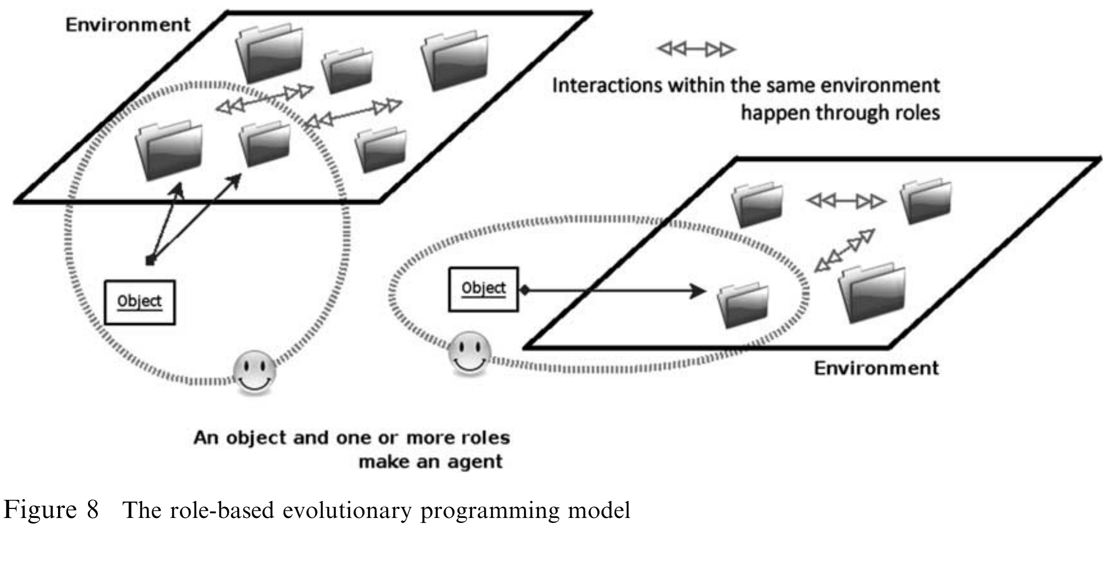
Environment
Interactions within the same environment happen through roles
Object
Object
Environment
An object and one or more roles make an agent

Figure 8 The role-based evolutionary programming model

interact with each other (criterion 1.2). In fact, roles permit the definition of relationships between different components, which can dynamically change by moving from one environment to another. RoleEP exploits the roles in particular at the implementation phase (criterion 1.3), where different functionalities can be implemented in a separate way, allowing a modular approach and an easy maintenance of the applications. Moreover, the capability of dynamically binding the role functions to objects (*creating agents*, in the RoleEP terminology) grants a high degree of openness (criterion 4) and flexibility, since agents can assume roles dynamically, depending on the runtime needs and situations.

3.8 Role-oriented programming environment

The ROPE (role-oriented programming environment) recognizes the importance of defining roles as *first-class* entities (Becht *et al.*, 1999). As other proposals, ROPE allows agents to assume roles in order to cooperate with each other.

ROPE focuses in particular on the concept of *cooperation processes* (CPs), used to model cooperation among agents. A CP defines how the involved roles interact with each other, and it is described by a formal notation (criterion 2) derived from the Petri nets (Becht *et al.*, 1998). CPs are considered as first class entities as well as roles.

The ROPE proposal exploits roles as interfaces between an agent and a CP, and defines a role as:

- a set of *required permissions*, which are the permissions an agent must have in order to be allowed to play a role;
- a set of *granted permissions*, which are the permissions an agent acquires by playing the role;
- a directed graph of *service invocations*, which defines how services can be exploited and exposed to other agents;
- a *state*, which defines the current information handled by the couple agent-role, and that changes during the agent life cycle depending on its interactions and cooperation with other agents and roles.

ROPE distinguishes itself from other proposals also by its definition of agent: an agent is an *entity consisting of a set of provided services*. Such services are made available to other agents, to carry out the task for which they cooperate. After assuming roles, agents can participate in a CP, where they make their services available to other agents, and exploit services of the others to fulfill a common task. Starting from the above concepts and definitions, it is possible to depict the ROPE model as in Figure 9.

ROPE provides a specification language in order to define the process cooperations, and a code generator that allows an automatic definition of a ‘ready to run’ code that matches the defined process and its interaction rules. The result is then executed within a ROPE Engine that applies the interaction rules to the running processes.

Thanks to the defined coordination language, which is a special type of Petri’s Net, ROPE allows the developer to clearly focus on the design phase (criterion 1.2), while the code generator makes ROPE an implementation focused approach (criterion 1.3; see Figure 10).

ROPE allows significant decoupling of the definition of the agents from the structure of CPs. This grants openness (criterion 4) and flexibility in the development of cooperative applications, and permits the reuse of solutions for different situations.

3.9 Tractable role-based agent prototype for concurrent navigation systems

The tractable role-based agent prototype for concurrent navigation systems (TRANS; Fournier *et al.*, 2003) is a Java system that models ship navigation using agents. In particular, agents either represent ships (mobile) or natural entities (fixed), such as obstacles. TRANS exploits roles to better model real situations: each mobile agent (i.e. a ship) plays one or more roles (e.g. passenger boat, tank, etc.). A role, which is defined as a single object, when applied to an agent, defines the *behavior* of the agent itself, and thus of the ship it represents, creating a modularized environment.

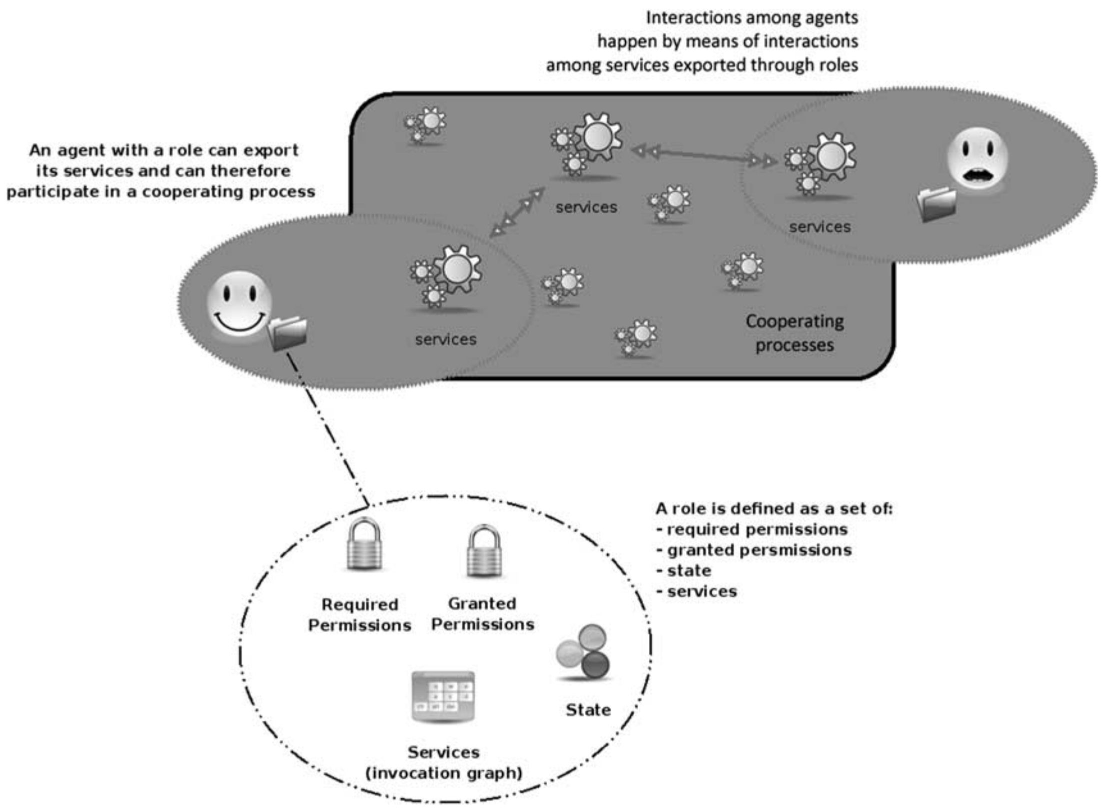
Interactions among agents
happen by means of interactions
among services exported through roles
An agent with a role can export
its services and can therefore
participate in a cooperating process
services
services
services
Cooperating
processes
A role is defined as a set of:
- required permissions
- granted persmissions
- state
- services
Required
Permissions
Granted
Permissions
State
Services
(invocation graph)

Figure 9 The role-oriented programming environment model

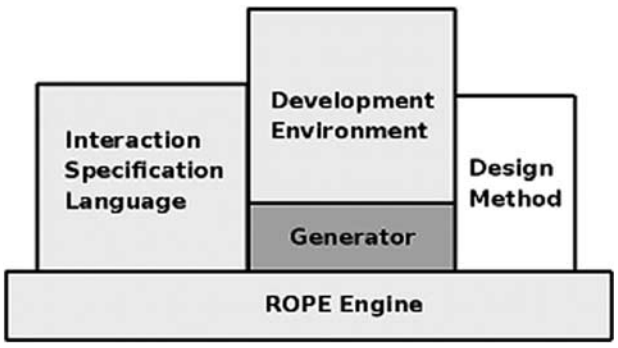

Figure 10 role-oriented programming environment development components

There are several attributes of a role in TRANS:

- *Priority*: since an agent can assume several roles during its life, TRANS decides the assumption sequence through a priority; thus, the agent is forced to assume, among several available roles, the one with the highest priority;
- *Permanence*: a role can be *permanent*, if the agent will never release it, or *temporary*, if the agent can release it;
- *Incompatibility*: it is possible to define two roles as incompatible, meaning that an agent cannot assume both at the same time;
- *Exclusivity*: a role can be considered *exclusive* if an agent cannot assume other roles while playing the former.

In TRANS, interactions between agents and the environment (i.e. other agents) are realized through roles. It is important to note that TRANS organizes agents in groups, and roles are

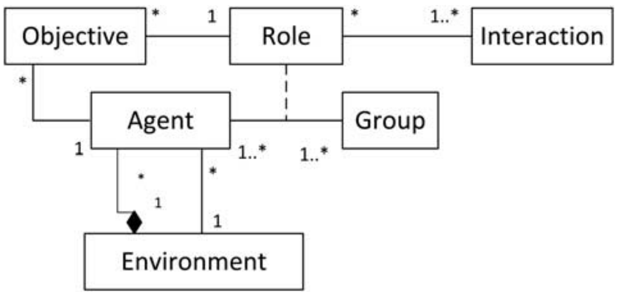

Figure 11 A Portion of the TRANS UML class diagram

considered as reification of the association agent-group (see Figure 11). This also imposes that an agent can play only one role per group to which it belongs. Figure 11 emphasizes also that an agent should choose a role to play depending on its objectives.

Thanks to the above concepts, and to a formal notation (criterion 2), the developer can define, at a high level, how the agents will act/interact in the TRANS system. This will provide a good behavior model at the design level (criterion 1.2), leading to a simpler implementation (criterion 1.3) that is supported by TRANS and relies on Java, by means of an independent class for each role. There is no particular mechanism that drives the role assumption/release: an agent that wants to dynamically play a role has simply to instantiate an object of the role class and to use it as a normal object (criterion 3). The release of the role corresponds to the destruction of the role object itself.

The fact that a role is a consequence of the association agent-group can limit the applicability of the proposal itself. In fact, since roles are chosen depending on the group to which an agent belongs, the scenario is not very dynamic, and cannot evolve. For example, it could happen that an agent cannot be classified in present groups, and thus it cannot even play a role to achieve its objectives. It would probably be better to group agents by their roles, and not vice versa. This is probably a lack that comes from the system objectives: being studied to simulate ship navigations, the system is too tied to this application scenario, and thus it is not general enough to be applied to generic scenarios.

3.10 Tasks and roles in a unified coordination environment

The tasks and roles in a unified coordination environment (TRUCE) is a script-based language framework for the coordination of agents (Jamison & Lea, 1999), which aims to overcome problems related to *adaptability*, *heterogeneity* and *concurrency*. This leads TRUCE authors to exploit roles in order to separate the application (i.e. algorithmic) issues from the coordination (i.e. interaction) ones. Thanks to this separation, the coordination rules can be embedded into roles without worrying about agents that are going to use them.

TRUCE relies on the Agency-based Collaboration Architecture for Cooperating Intelligent Agents (ACACIA) environment (Jamison, 1996), even if it is not strictly bound to it. ACACIA provides for the coordination services exploited by agents written in TRUCE. This proposal defines the following concepts (see Figure 12):

- *Collaboration group* is a set of possibly heterogeneous agents that cooperate to solve a common problem. Each group is governed by a set of coordination protocols, used to make the coordination possible.
- *Role* describes a *view* of an agent, hiding some details about it, thus leaving other agents viewing only the properties concerned by the assumed role.

The TRUCE proposal does not address the analysis and design phases, focusing only on the implementation phase (criterion 1.3). In fact, coordination is a consequence of the interpretation of *coordination scripts*, which are executed by agents. Each script contains appropriate tags, which

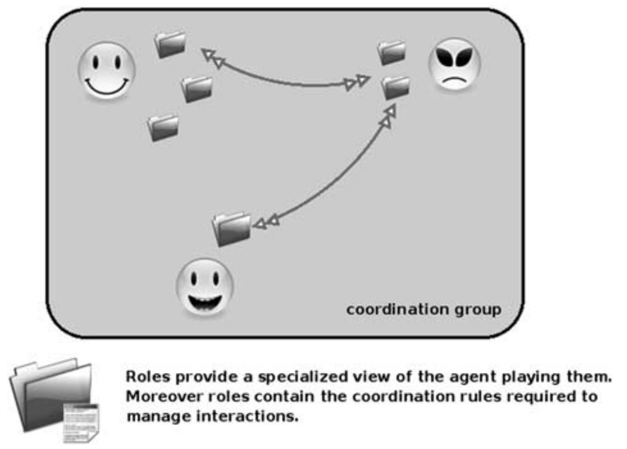
coordination group
Roles provide a specialized view of the agent playing them. Moreover roles contain the coordination rules required to manage interactions.

Figure 12 Agent interacting through their partner view/role

specify which role has to execute the tagged code, exploiting concretely the concept of role during a concurrent execution of coordination scripts by agents belonging to the same group.

TRUCE allows coordinating groups to exploit different protocols, and this makes the model flexible and adaptable to several application scenarios. Moreover, this proposal addresses the separation of coordination and computational phases, granting the reusability of protocols and decentralization of the coordination process. Finally, since it is possible for an agent to assume several roles, TRUCE allows an agent to concurrently participate in different coordination and computational scenarios, also making possible a binding among them (criterion 4).

3.11 Yu and Schmid's Proposal

Yu and Schmid (1999) exploit roles assigned to agents to manage workflow processes. They model a role as a collection of *rights* (activities an agent is permitted to perform on a set of resources) and *duties* (activities an agent must perform). More particularly, this proposal defines a role through the following attributes (see Figure 13):

- *Name*, identifying the role;
- *Description*, listing the functions of the role;
- *Goals* for which the agent playing this role is responsible;
- *Qualifications*, pre-conditions or skills required to achieve the goals;
- *Relationships*, describing the relations with other roles in a workflow;
- *Obligations*, listing the duties of the role;
- *Concurrency constraints*, specifying the synchronization and the parallelism of obligations;
- *Permissions*, specifying what an agent playing this role can or cannot do;
- *Protocols* in which the role plays a part to achieve the goals.

An interesting issue of this proposal is that it aims to cover different phases of the application development, proposing a *role-based analysis* phase, an *agent-oriented design* phase and an *agent-oriented implementation* phase.

The *role-based analysis* phase (criterion 1.1) is divided into three steps. The *first step* identifies the goals of the workflow, and divides them into sub-goals, which are associated with activities. Activities can be elementary (they cannot be further divided) and complex (composed of sub-activities). In the *second step*, the roles are determined and specified. Activities are not assigned directly to given agents, but are in charge of a role, which is a prototypical function of an agent in a workflow. Roles are defined as collections of elementary roles, which in turn are collections of complex tasks or obligations; also, aggregate roles are defined, which are collections of roles and coincide with social positions inside the organization. The *third step* relates to the specification of

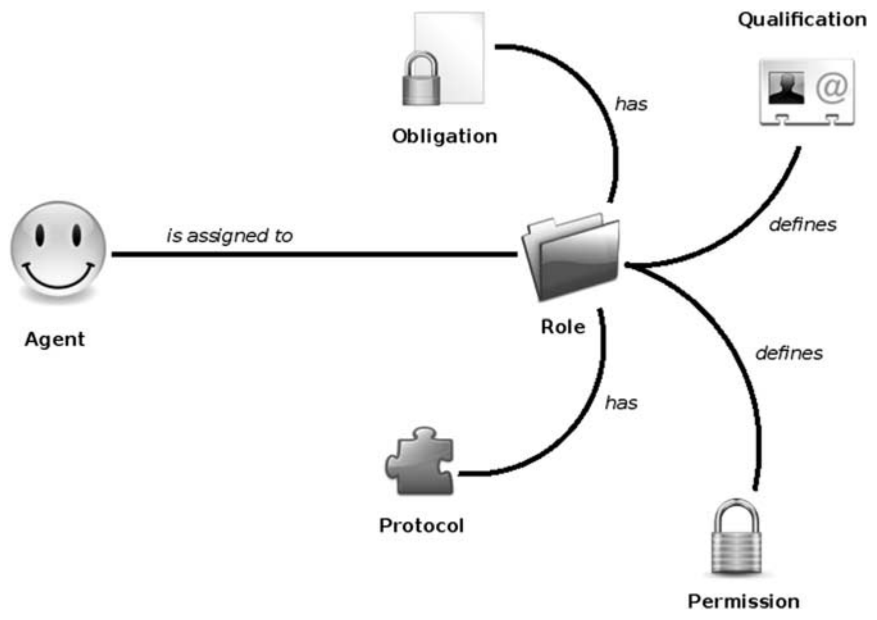

Figure 13 Yu and Schmid proposal

protocols for inter-role interactions. They can be specified by formal description techniques—the authors exploit Finite State Machines formalism—(criterion 2) to keep consistency, which is mapped into event condition action rules. The interaction messages in the transition rules must be detailed in this step, and the messages exchanged are represented as performatives (Knowledge Query Manipulation Language speech acts). In our opinion, such a constraint is too strict at the analysis phase, and a more abstract communication mechanism should be allowed, to leave the choice of the ACL at the implementation phase.

The *agent-oriented design* phase (criterion 1.2) aims at identifying agent types and assigning roles to them. Since this proposal is related to workflow, the authors identify some specific agents in the field, but we think that this phase can be adapted to different areas. This phase also concerns the definition of further details (not related to roles and interaction protocols) of the agents.

The *agent-oriented implementation* phase is however not particularly considered by the authors, who focus on the first two phases. They suggest exploiting existent agent platforms; however, these do not necessarily implement role concepts (criterion 1.3 partially).

Three advantages emerge from adopting roles for modeling agent workflow. First, they enable separation between agents and the definition of the process in which they are involved. Second, they provide a high-level abstraction of agents, describing what are the functions and the relationships needed by the system. Third, the role-based analysis well suits the agent-oriented design phase, helping and supporting the application developers in their work.

The fact that this proposal focuses on the workflow management makes it quite close, but in our opinion there are some interesting cues that can be exploited in a wider range of application areas.

3.12 Zhu and Zhou's proposal

Zhu and Zhou (2006) describe a role model that is tied to both the computers and humans involved in collaborations, and in particular tries to provide help to humans in computer-supported collaborations. This model defines a role as a set of *responsibilities* and *capabilities* a human holds, providing then other concepts:

- *Object* is an instance of a class, as in the pure object-oriented paradigm. An object can be used to express everything in the collaborative system;

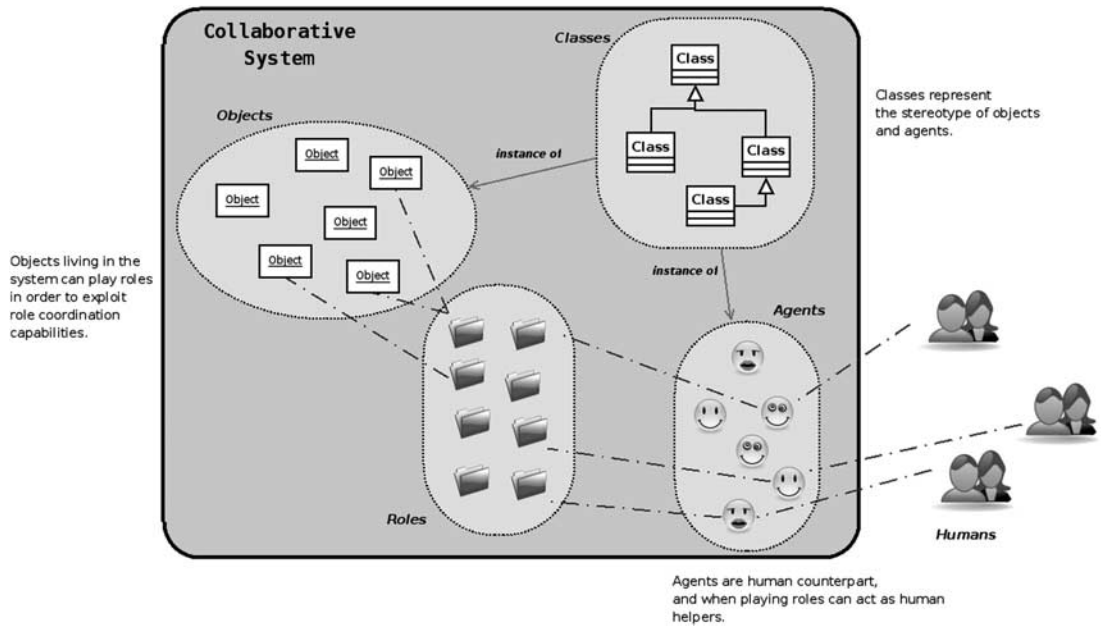
Collaborative System
Classes
Class
Class
Class
Class
Classes represent the stereotype of objects and agents.
Objects
Object
Object
Object
Object
Object
Object
instance of
instance of
Objects living in the system can play roles in order to exploit role coordination capabilities.
Agents
Roles
Humans
Agents are human counterpart, and when playing roles can act as human helpers.

Figure 14 A possible scenario in Zhu and Zhou's collaborative system

- *Agent* is a special kind of object, which represents a human counterpart;
- *Group* is a set of correlated agents;
- *Message* is a command sent from a component of the system to another one, even the human user (or her agent counterpart);
- *Role agent* is a particular kind of agent tied to a specific role. Role agents act as human helpers, providing an interface between a human user and the collaborative system.

These concepts are exploited in both the analysis (criterion 1.1) and design (criterion 1.2) phases of development.

The key to this proposal is the concept of *role agent*, which is an agent with a role attached. An agent that wants to participate in collaboration must own at least one role. Role agents should help the human during the collaboration, for example, helping her to send messages to other agents or entities in the collaborative system (e.g. all agents in the same group). Role agents can change their role at runtime (criterion 3), leading to a dynamic environment and promoting the use of specific roles for specific tasks. Each human must log in the system with a particular role, so she is bound to a role agent, which produces objects as result of the collaboration and/or human wills. Using the above concepts, it is possible to model the collaborative system as in Figure 14.

This proposal exploits a formal notation (criterion 2), which emphasizes that a role is defined not as an object but as a set of messages (both outgoing and incoming) that the role agent can send/receive during the collaboration. Therefore, a role agent is an agent that is able to exchange particular messages, with a content dependent on the collaboration the agent is involved in.

The ECARGO system is not a tool to support the implementation phase, but only an example of implementation (criterion 1.3 partially) of the proposed model (Seguin & Zhu, 2006).

The proposal proposed by Zhu and Zhou takes care of human activities, but from a computer science point of view this can be brought down to interactions among agents that represent human counterparts. Most of the concepts proposed in this proposal, such as groups and messages, are not new, and the definition of roles as sets of messages is not really flexible. Nevertheless, this proposal well reflects a real situation, and being based on a formal notation, can help designers and developers planning the system.

3.13 Further considerations

Before comparing all the presented proposals and suggesting how to choose the most suitable ones for specific situations, it might be helpful to summarize other characteristics of the above proposals.

All proposals use roles to gain *flexibility* in the development process, even if at different levels. Most of them exploit roles to clearly separate issues related to the execution of the agent, such as mobility, from issues related to its sociality, such as interaction protocols. In addition, roles are seen as a way to achieve a high degree of code reusability, since almost all proposals allow the use of the same role by different agents. A concept present in all proposals is that roles are useful for collaborative environments, where agents have a common task to achieve and collaborate to do that.

All proposals recognize that an agent can play *multiple roles* at the same time, but it is only in RoleEP that an agent cannot be defined without a role (see Section 3.7). In other words, most of the proposals enable interactions also between agents that are not playing roles, while RoleEP considers only interacting agents as entities that *must* play at least one role. This can clash with other definitions of agents (Luck *et al.*, 2003), creating a mess around the concept of agent.

A few proposals (AALAADIN, TRANS and Zhu and Zhou's version—see Sections 3.1, 3.9 and 3.12) introduce the notion of *group*, which, even if defined in different ways, is a set of related agents, where the term 'related' is used to express the fact that agents have a common task. Groups are very useful, in particular during the analysis phase, since they define 'sub-environments' where agents agree on communication protocols and where available roles are coherent with each other.

BRAIN, Fasli's proposal and Gaia (Sections 3.2, 3.3 and 3.4) explicitly define a role also as a set of *duties*, which means that agents playing a role should behave according to it. This can seem obvious, since an agent playing a specific role *should* behave according to it, exploiting the role services/capabilities/rights. Instead, the definition of *duties* is stronger, since it requires that the agent playing such a role must act in a specific way depending on external events, which means on social events. In other words, the agent is obligated to act in a specific way depending on the evolution of the social scenario (e.g. requests of services from other agents).

All proposals define a role as a *set of rights*, even if not explicitly. In fact, those proposals that do not use explicitly the term 'right' define security rules or issues related to each role, so that rights (i.e. the right to do something) coincide with permissions (see, e.g. Gaia in Section 3.4 and ROPE in Section 3.8).

Only two proposals define roles also as *views* of agents playing them (BRAIN and TRANS—see Sections 3.2 and 3.9). We believe that this can be an interesting feature, since a role assumption/dismiss should also change the way agents recognize each other. For example, if an agent is playing the role of 'tank' in TRANS, other agents should see it as a 'tank', which means that they should *understand* that they can ask it for fuel. Here, the term 'understand' means that agents should be able to recognize the role played by other agents without explicitly asking them for it.

An important characteristic, in our opinion, is the definition of roles as *first-class entities*, which is not stressed by all presented proposals. In fact, some proposals give a definition of the concept of role that is too generic and too far from implementation issues.

It is important to note that there is no proposal that defines a way, or provides tools, to know which roles should be defined. In fact, giving a set of interactions, how many roles should a developer provide to cover them? Providing developers with such tools will help them to focus on the right number of components (i.e. roles) required by the application, and concentrate their efforts in designing/implementing them.

Most of the presented proposals emphasize the difference between 'normal' entities and 'social' ones (RICA—Section 3.6—recognizes even a communication entity as a specialization of a social one). In our experience, this specialization is useless most of the time, since the separation edge between social and unsocial entities is really thin. In fact, it may happen that an unsocial entity (such as an unsocial role) produces results that will be used later in a social interaction, and thus the above entity could be redefined as social. Furthermore, the definition of 'social' seems to be quite personal, and thus it is better to let the developer define what she finds to be 'social' and what is not.

Another important aspect to note, which is tied to the implementation level, is how proposals *maintain* roles. Since agents could be mobile, and roles can help them interact in all different scenarios where they are moving, how should roles be distributed among all platforms/sites? No proposal focuses on it, and the only one that provides a minimal support to this problem is BRAIN, which, thanks to the concept of role descriptors (Cabri *et al.*, 2003a), helps role upgrading/refactoring in a way that is transparent to agents.

Among the considered ones, only one proposal (TRANS, Section 3.9) gives the notion of *priority*, which defines a dependence chain among roles. Thanks to priority, it is possible to drive the role assumption process, defining which role must be assumed before others, leading to a more predictable agent behavior. The BRAIN framework proposes something similar, even if more complex, because the authors introduced the concept of *role evolution*, which defines dependence chains among roles (Cabri *et al.*, 2004a).

Finally, another form of *interoperability* can be the capability to mix the role analysis, design and implementation among different proposals; thus, for example, a designer can use the analysis phase of one proposal, while during development, it can use the development phase of another proposal. In this sense, the considered proposals seem not interoperable, at least in an easy way.

4 Comparison

This section reports the comparison among all presented proposals based on the evaluation criteria presented in Section 2. Then we try to suggest how to choose the most suitable proposal(s) for given purposes.

4.1 Comparison of the proposals

This section summarizes all the evaluated criteria in the presented proposals. Table 1 shows, for each evaluated proposal, the support for analysis, design, implementation and characteristics such as the capability of using roles at runtime, the presence of a formal notation in the role definition, the openness of the proposal and its interoperability with other systems/proposals. In such a table, the symbol ✓ means that there is full support, while ☑ means partial support (i.e. without roles), and – means that the characteristic is absent. With regard to the Formal notation column, we report the names of some notations that have been adopted by the proposals, but not designed explicitly for them.

With regard to the support for design, analysis and implementation only, Figure 15 shows a different view of the presented proposals: each proposal is presented as a star in the circle, and its position depends on how much it is implementation, design or analysis prone. Of course, proposals that present a balanced support for all phases are those near the center of the circle, while proposals prone to only one development phase are near to the corresponding axis. Figure 15 details better than Table 1 a direct comparison of the various proposals on the first three aspects (implementation, analysis and design). As readers can see, for example, BRAIN is closer to implementation than AALAADIN, even if they support all phases; similar considerations can be made for other proposals depending on the key aspect developers are interested in.

There are several proposals that do not support the implementation phase with roles, which means, for example, that they do not define roles as entities themselves, but as abstract concepts. This is very similar to the human world, where a role is an abstract idea, but can produce incoherent implementations in the agent world. Furthermore, such proposals cannot apply their role model to all development phases, leading to a fragmented solution. As it is possible to note, only proposals that completely support the implementation phase can provide a dynamic role assumption/release. Only BRAIN, Kendall and RoleEP proposals cover such a dynamism, which is the capability to assume/release roles at runtime without static dependencies (i.e. dependencies defined in the code). This is a really important feature, since it is the only one that can make agents suitable for dynamic applications (as those for the Internet) granting a *real adaptability* to environment changes.

A formal notation, which is exploited from most proposals, helps in the analysis/design phase, but it would be friendlier if managed in a graphical way or used in connection with a visual notation, such as AUML[4]. In fact, a notation can be harder to manage than an AUML diagram, while the UML standard diagram cannot provide a visual representation of the concept of role if the proposal does not define it as an entity itself. No proposal seems to use the AUML notation, which is explicitly conceived for agents and roles. The fact that the presented proposals do not use standard instruments such as AUML, even preferring their own ones, causes a heterogeneity that cannot help developers migrating from one proposal to another. Furthermore, only the RICA theory has an implementation that takes into account the merging with an FIPA standard system, while all other proposals, even if their implementation is adaptable to the existing platform, do not.

It is worth noting that, as shown in Figure 15, there is no proposal close to the design axis, meaning that there is no proposal that is tailored to design. In other words, proposals are generally practical (close to implementation) or abstract (favoring analysis), but never too schema-oriented.

With regard to those proposals near to the center of Figure 15, we point out that they are quite recent proposals, and so it is natural that they are more balanced and general than others, since they have been built upon other proposal experiences.

Table 1 Comparison of the proposals

	Support for (criterion 1)			Formal notation	Dynamic roles allocation	Openness	Interoperability
Proposal	Analysis	Design	Implementation	(criterion 2)	(criterion 3)	(criterion 4)	(criterion 5)
AALAADIN	✓	✓	☑	–	–	✓	✓
BRAIN	✓	✓	✓	✓	✓	✓	✓
Fasli	✓	–	–	✓	–	–	–
Gaia	✓	☑	–	✓ FUSION-based	–	–	–
Kendall (AOP)	✓	✓	✓	–	✓	–	–
RICA (RICA-J)	✓	✓	✓	✓	✓	✓	✓
RoleEP	–	✓	✓	–	✓	✓	–
ROPE	–	✓	✓	✓ Petri net[5]	✓	✓	–
TRANS	–	✓	✓	✓	✓	–	–
TRUCE	–	–	✓	–	–	✓	–
Yu and Schmid	✓	✓	☑	✓ Finite state machines[6]	✓	–	–
Zhu and Zhou	✓	✓	☑	✓	✓	–	–

4.2 Choosing the most suitable proposal

As written in the introduction, the aim of this survey is not to find out which proposal currently represents the best, but rather to help developers orient to choose the proposal that is better suitable for their purposes. For example, BRAIN and RICA can be considered complete proposals, even if their API could be more difficult to use than other proposals. Similarly, other proposals can have specific advantages related to developer purposes, which will be pointed out in the following to help the developers' choice.

If the application must be built rapidly, or if it is very simple and thus there is no need for a deep analysis, developers should choose Kendall' proposal or TRUCE or TRANS. These proposals grant separation of concerns, even if they are very practical.

[4] `http://www.auml.org`

[5] Only for cooperation processes.

[6] Only for interaction protocols.

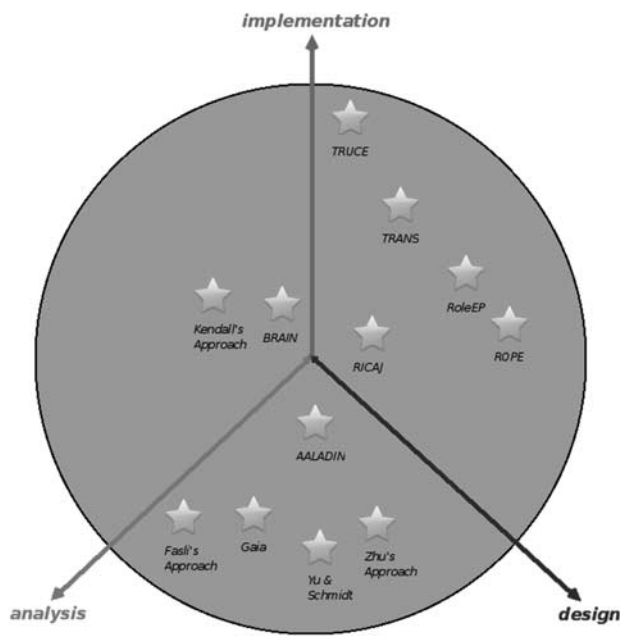

Figure 15 Proposals and their support for implementation, design and analysis

In situations where the application requires a deep analysis, AALAADIN, Gaia, Fasli's or Yu and Schmidt's proposals are the best, while Zhu and Zhou's proposal becomes really useful if the system also needs to interact with humans. As already written, there is no proposal that particularly focuses on the design phase, and thus the choice in these situations is equally distributed among all the proposals.

For situations that focus more on services than on all the agents' features, the ROPE proposal seems to be more appropriate.

Another important issue that can lead the choice of the proposal to use can be dynamism, which is an implementation factor and, in this direction, the best proposals are BRAIN and RoleEP.

5 Conclusions

This paper has presented a survey of different proposals based on roles for agents. We have chosen to present only the most popular ones, after evaluating a lot of proposals, in order to emphasize their typical characteristics and behaviors. As already mentioned, we have not considered proposals that rely on roles but do not have roles as their main focus, or proposals that are too abstract, too simple or too specific.

What we found from this survey is that roles provide advantages in the development of agent-based applications, in particular in engineering interactions. The main advantage of using roles is the separation of concerns that they enable, which can be useful in all the development phases. Such a separation allows developers to focus on a given aspect of the application, reducing complexity in the development process. From the conceptual point of view, roles turn out to be very suitable for agent-based approaches, since they promote an organizational view of the systems. Therefore, an analysis based on roles helps in understanding the main functions of the systems, and can be easily translated into an agent-based design model.

A common limitation of the presented proposals is the lack of support for development during all phases, without leading capabilities such as dynamism, openness and interoperability. Furthermore, we believe that the next generation of role approaches should provide developers with tools more powerful than notations; thus, developers can be helped to understand exactly which roles are required and how to implement them in an easy way.

We believe that all role proposals should move in the direction of interoperability not only at a platform level, but also between the different development phases, because only in this way can developers become free to exploit different proposals, each one with its own advantages and drawbacks, at different phases. This will lead to development suitable for all agent programmers.

Acknowledgments

This study was partially supported by the Italian MiUR in the frame of the PRIN project 'MenSA—agent-oriented methodologies: engineering of interactions and relationship with the infrastructures'. The authors would also like to thank the anonymous referees for their precious suggestions.

References

Baumer, D., Ritchie, D., Siberski, W. & Wulf, M. 1997. The role object pattern. In *Proceedings of the 4th Pattern Languages of Programming Conference (PLoP)*, Monticello, Illinois, USA.

Becht, M., Muscholl, M. & Levi, P. 1998. Transformable multi-agent systems: a specification language for cooperation processes. In *Proceedings of the World Automation Congress, ISOMA'98*, Anchorage, Alaska, USA.

Becht, M., Gurzki, T., Klarmann, J. & Muscholl, M. 1999. ROPE: role oriented programming environment for multiagent systems. In *Proceedings of the Fourth IFCIS Conference on Cooperative Information Systems (CoopIS'99)*, Edinburgh, Scotland.

Biddle, B. J. 1979. *Role Theory: Expectations, Identities, and Behaviors*. Academic Press.

Cabri, G., Leonardi, L. & Zambonelli, F. 2000. MARS: a programmable coordination architecture for mobile agents. *IEEE Internet Computing* **4**(4), 26–35.

Cabri, G., Leonardi, L. & Zambonelli, F. 2003a. BRAIN: a framework for flexible role-based interactions in multiagent systems. In *Proceedings of the 2003 Conference on Cooperative Information Systems (CoopIS)*, Catania, Italy.

Cabri, G., Leonardi, L. & Zambonelli, F. 2003b. Implementing role-based interactions for internet agents. In *Proceedings of the 2003 International Symposium on Applications and the Internet* (*SAINT*), Orlando, Florida, USA.

Cabri, G., Ferrari, L. & Leonardi, L. 2004a. Agent roles in the BRAIN framework: rethinking agent roles. In *Proceedings of the 2004 IEEE Systems, Man and Cybernetics Conference, Session on "Role-based Collaboration"*, The Hague, The Netherlands.

Cabri, G., Ferrari, L. & Zambonelli, F. 2004b. Role-based approaches for engineering interactions in large-scale multi-agent systems. In *Software Engineering for Multi-agent Systems II*, Pereira de Lucena, C. J., Garcia, A. F., Romanovsky, A., Castro, J. & Alencar, P. (eds). Lecture Notes in Computer Science **2940**, 243–263. Springer-Verlag.

Communications of the ACM. 2001. Special issue on aspect oriented programming, **44**(10), ACM Press, October 2001.

Castelfranchi, C. 1995. Commitments: from individual intentions to groups and organizations. In *Proceedings of the First ICMAS Conference*, San Francisco, California, USA, 41–48.

Demsky, B. & Rinard, M. 2002. Role-based exploration of object-oriented programs. In *Proceedings of the International Conference on Software Engineering 2002*, Orlando, Florida, USA.

Fasli, M. 2003. Social interactions in multi-agent systems: a formal approach. In *Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology (IAT 2003)*, IEEE Press, 240–247.

Ferber, J. & Gutknecht, O. 1998. AALAADIN: a meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS'98)*, Paris, France.

Ferber, J., Gutknecht, O. & Michel, F. 2004. From agents to organizations: an organizational view of multiagent systems. In *Agent-Oriented Software Engineering (AOSE) IV*, Giorgini, P., Müller, J. & Odell, J. (eds). Lecture Notes in Computer Science **2935**, 214–230. Springer, Melbourne, July 2003.

Fournier, S., Brocarei, D., Devogele, T. & Claramunt, C. 2003. TRANS: a tractable role-based agent prototype for concurrent navigation systems. In *Proceedings of the First European Workshop on Multi-Agent Systems (EUMAS)*, Oxford, UK.

Fowler, M. 1997. *Dealing with Roles*. `http://martinfowler.com/apsupp/roles.pdf`

Gutknecht, O. & Ferber, J. 2000. The MadKit agent platform architecture. In *Proceedings of the 1st Workshop on Infrastructure for Scalable Multi-Agent Systems*, Barcelona (E).

Hirsh, B., Fisher, M. & Ghidini, C. 2003. Programming group computations. In *The Proceedings of the First European Workshop on Multi-Agent System (EUMAS)*, Oxford, UK.

Jamison, W. 1996. ACACIA: an agency based collaboration framework for het-erogeneous multiagent systems. In *Multiagent Systems Methodologies and Applications*, Lecture Notes in Artificial Intelligence **1286**, 76–91. Springer-Verlag.

Jamison, W. & Lea, D. 1999. TRUCE: agent coordination through concurrent interpretation of role-based protocols. In *Proceedings of Coordination 99*, Amsterdam, The Netherlands.

Kendall, E. A. 2000. Role modelling for agent systems analysis, design and implementation. *IEEE Concurrency* **8**(2), 34–41.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J. M. & Irwin, J. 1997. Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, Lecture Notes in Computer Science **1241**, 220–242. Finland.

Kristensen, B. B. & Østerbye, K. 1996. Roles: conceptual abstraction theory & practical language issues. *Special Issue of Theory and Practice of Object Systems on Subjectivity in Object-Oriented Systems* **2**(3), 143–160.

Luck, M., McBurney, P. & Preist, C. 2003. *Agent Technology: Enabling Next Generation Computing — A Roadmap for Agent Based Computing*, AgentLink. `http://www.agentlink.org/roadmap`

Odell, J. J., Van Dyke Parunak, H. & Fleischer, M. 2003. The role of roles in designing effective agent organizations. Lecture Notes in Computer Science **2603**, 27–38. Springer.

Omicini, A., Ricci, A. & Ossowski, S. 2003. Rethinking MAS infrastructure based on activity theory. In *Proceedings of the First European Workshop on Multi-Agent System (EUMAS)*, Oxford, UK.

Sadek, M. D. 1991. Dialogue acts are rational plans. In *Proceedings of the ESCA/ETRW Workshop on the Structure of Multimodal Dialogue*, Maratea, Italy.

Sandhu, R. S., Coyne, E. J., FeinStein, H. L. & Youman, C. E. 1996. Role-based access control models. *IEEE Computer* **20**(2), 38–47.

Seguin, P. & Zhu, H. 2006. Implementing a tool for role-based collaboration. In *Canadian Conference on Electrical and Computer Engineering*, Ottawa, Canada, 2428–2431.

Serrano, J. M. & Ossowski, S. 2004. On the impact of agent communicative languages on the implementation of agent systems. In *Cooperative Information Agents VIII*, Klush, M., Ossowski, S., Kashyap, V. & Unland, R. (eds). Lecture Notes in Artificial Intelligence, ISSN 0302-9743, 92–106. Springer.

Shneiderman, B. & Plaisant, C. 1994. The future of graphic user interfaces: personal role manager. In *Proceedings of the Conference on People and Computers IX*, Glasgow, UK.

Sierra, C., Rodriguez-Aguilar, J. A., Noriega, P., Esteva, M. & Arcos, J. L. 2004. Engineering multi-agent systems as electronic institutions. *European Journal for the Informatics Professional* **4**, 33–39.

Tripathi, A., Ahmed, T., Kumar, R. & Jaman, S. 2002. Design of a plicy-driven middleware for secure distributed collaboration. In *Proceedings of the 22nd International Conference on Distributed Computing System* (*ICDCS*), Vienna (A).

Ubayashi, N. & Tamai, T. 2000. RoleEP: role based evolutionary programming for cooperative mobile agent applications. In *Proceedings of the International Symposium on Principles of Software Evolution*, Kanazawa, Japan.

Wooldridge, M. 2002. *An Introduction to Multi-agent Systems*. John Wiley and Sons.

Yu, L. & Schmid, B. F. 1999. A conceptual framework for agent-oriented and role-based workflow modeling. In *Proceedings of the 1st International Workshop on Agent-Oriented Information Systems*, Wagner, G. & Yu, E. (eds). MIT Press, Heidelberg.

Zambonelli, F., Jennings, N. & Wooldridge, M. 2001. Organizational rules as an abstraction for the analysis and design of multi-agent systems. *Journal of Knowledge and Software Engineering* **11**(3), 303–328.

Zambonelli, F., Jennings, N. & Wooldridge, M. 2003. Developing multiagent systems: the Gaia methodology. *ACM Transactions on Software Engineering and Methodology* **12**(3), 317–370.

Zhu, H. & Zhou, M. C. 2006. Role-based collaborations and their Kernel mechanisms. *IEEE Transactions on Systems, Man and Cybernetics, Part C* **36**(4), 578–589.