Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 7, Number 1, pp. 37–47. http://www.scpe.org © 2006 SWPS

AGENT COMPOSITION VIA ROLE-BASED INFRASTRUCTURES

GIACOMO CABRI*

Abstract. Software agents represent an interesting paradigm to approach complex and distributed systems. Their sociality enables to build multiagent systems, where different agents interact to pursue their goals. Multiagent systems can involve both cooperative and competitive agents. In both cases, the composition of different agents is an issue that must be faced by developers. In this paper, we propose to build infrastructures based on roles, which are abstractions that enable the composition of different agents in an open scenario. Some concrete examples are provided to support our proposal.

Key words. agents, roles, multiagent systems, interaction

1. Introduction.

With no doubt software agents have been proposing as an innovative paradigm for some years, and we envision a digital world populated by them to support users belonging to the human world. In fact, they are able to perform tasks on behalf of users, due to their main features—*autonomy, proactiveness, reactivity* and *sociality* [19]. In addition, complex applications can be divided into smaller and simpler tasks, each one delegated to one agent [18]. This leads to systems composed of several agents, called *multiagent systems*, where agents interact and coordinate to carry out a common goal. Going further, in a wide open scenario the social behaviour of the agents implies interactions not only between agents cooperating in one application, but also between agents of different applications, which may have a competitive behaviour, to gain the use of resources [12]. The feature of *mobility* [20], enhancing the autonomy of agents, implies further advantages. Generally, mobile agents can save bandwidth by moving locally to the environments where the resources are located, and do not rely on continuous network connections. Users are not required to be connected to the network continuously: they can send their agents, disconnect, and then reconnect when the agents have carried out their tasks to retrieve them.

However, building multiagent systems is not so easy, because they require a careful and effective composition of different agents, to carry out their tasks; the presence of mobile agents may make the scenario even more complex, since agents can enter and exit an application context dynamically. Composition means not only specifying which agents take part in a given application context, but also taking into account the rules for the interactions between agents and between agents and environments. Developers must face these issues during the entire development process. This paper proposes to compose agents by means of *infrastructures* based on the concept of *role*. A role can be defined as a stereotype of behaviour, and is exploited in particular in a group or organization of entities, each one exhibiting a specific behaviour inside the group. Real life proposes lots of examples where people play roles and the concept of role has been exploited in the Object-Oriented field to design complex applications, in the agent area, and in other proposals related to computer science in general. We show that roles can be exploited to build infrastructures that are useful abstractions to compose different agents in a multiagent system. Such abstractions can then be implemented exploiting a role-based system.

The paper is organized as follows. Section 2 introduces the concept of role for the agents. Section 3 presents the definition of infrastructures based on roles. Section 4 introduce the RoleX system, which can be exploited for a possible implementation. Section 5 shows some examples, giving more details about the implementation. Section 6 reports some related work. Finally, Section 7 concludes the paper and sketches some future work.

2. Roles.

One of the most important features of the agents is *sociality*, thanks which complex applications can be built on the base of the interactions between autonomous components (the agents themselves). In this context, it is important that application developers face the engineering of the interactions between agents with appropriate models and tools. To this purpose, a concept that seems to be suitable is *role*. The Oxford Dictionary reports: "*Role: noun 1. an actor's part in a play, film, etc. 2. a person's or thing's function in a particular situation.*" [23]. In describing patterns, Fowler says that roles are "some common behaviour of entities" that "do not have the same behaviour" [17], and points out that the isolation of such common behaviour can simplify the design of applications. Roles have been already exploited in Object Oriented approaches, where a role is applied to an object in order to change its capabilities and behaviour. Other approaches promote roles

*Dipartimento di Ingegneria dell'Informazione, Universita' di Modena e Reggio Emilia, via Vignolese, 905, 41100 Modena, Italy (cabri.giacomo@unimore.it).

as views of a particular object or entity [2], stressing the similarity between roles in computer programs and those in the real life.

Roles have been applied to agents promoting the reuse of solutions and making the definition of interaction scenarios easier. Roles allow not only the agent developers/designers to model the execution environment, but also allow agents to actively "feel" the environment itself. In other words, roles allow the developer and its agents to percept in the same way the execution environment. Roles can also be exploited to model and manage agent interactions [9], embedding all the capabilities (e.g., protocols, events) needed to handle a specific interaction. This is probably the most important meaning of roles: they allow and enable specific interactions, as well as social roles enable people, in the real world, to act in a certain way depending on the role they are playing in the society. There are some characteristics of roles that distinguish them from the concept of agent. The role is *temporary*, since an agent may play it in a well-defined period of time or in a well-defined context. Roles are *generic*, in the sense that they are not tightly bound to a specific application, but they express general properties that can be used in different applications. Finally, roles are *related to contexts*, which means that each environment can impose its own rules and can grant some local capabilities.

To better explain these concepts, we exploit an example in the software auction field, which will be taken up again later. Let us consider a software agent that is interested in acquiring a good or a service in a distributed environment. At runtime, after finding the good/service, it could discover that the good/service is put on sale by a specific auction house. To carry out its task, it can dynamically assume the role of bidder to attend the auction. In this case, the bidder is a behaviour that can be exhibited by different agents, of different applications and with different interests. But the features (or mechanisms) for bidding in a given auction house are independent of agents and can be embedded in the role of bidder. Moreover, different auction houses can provide different mechanisms and policies to rule the interactions. 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 examples of areas are *security*, in which we can recall the Role Based Access Control (RBAC) [24] that allows uncoupling between users and permissions, and the *Computer Supported Cooperative Work* (CSCW) [29], where roles grant adaptation and separation of duties. Also in the area of software development we can find approaches based on roles, especially in the *object-oriented programming* [13, 22], in *design patterns* [17], and in computer-to-human interfaces [27], which remark the advantages of role-based approaches.

Finally, it is important to note that roles can be exploited in a static or dynamic way. The static way imposes that roles are joined to an agent before its execution, while the dynamic way allows roles to be joined during at run-time, during the application execution.

3. Composing Agents via Role-based Infrastructures.

In the introduction, we state that multiagent systems are useful but must be developed carefully; in particular, agents of multiagent systems must be composed and their interactions must be designed. To support this job, we propose to build *infrastructures* as abstractions to compose different agents. These infrastructures are based on roles, and the key idea is that a set of roles determines an infrastructure that specifies which agents (or, better, which roles) take part in an application context and rules the corresponding interactions with other agents and with the environment. Such an infrastructure can be considered also as intermediate between applications and environments; in the following we consider the resources as part of the infrastructures, but actually this model can be adopted even in case of legacy software that remains outside the infrastructure. It is useful to identify some issues inside the infrastructures to simplify the design and to help the implementation and the deployment. We propose to model role-based infrastructures distinguishing three levels (see Figure 3.1):

- the *role level* contains the roles that agents can assume in the environment; each environment has its own set of admitted roles;
- the *policy & mechanism level* aims at defining the policies local to the environment and the mechanisms that implements the interaction among roles;
- finally, the *resource level* contains the local resources, such as information and services.

Above the role level we can find application agents that play roles. In this paper we disregard the internal constitution of agents, focusing on their external behaviours–captured by roles. We assume that agents are *network-aware*, which means that they perceive the network not as a whole flat environment, but instead as a set of environments, each one with given resources and services [8].

The role level can be considered as the interface of an environment for agents. To define the infrastructure, an administrator has to perform the following steps:

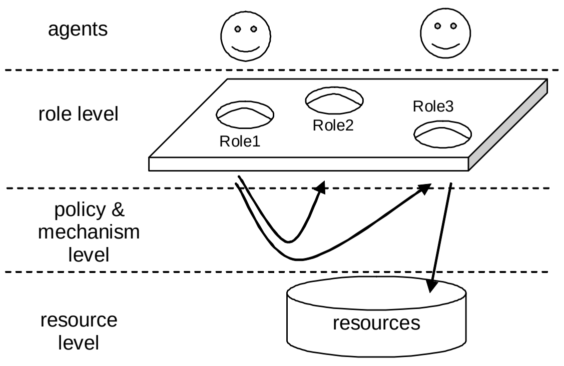

Fig. 3.1. *A role-based infrastructure*

1. to choose the *roles* her/his environment is going to support; often, this choice is implicit in the environment, as shown in the first example of the next section;
2. to define the *policies* by which the chosen roles can interact with each other or with the local resources.

This model of infrastructures leads to advantages at several phases of the application life cycle. At the *design stage*, roles permit separation of concerns, which allows the designer to concentrate on the single (interaction) issue, and the reuse of solutions. At the *development stage*, the reuse of roles permits to avoid the implementation of common (already implemented by someone else) functionalities. At the *runtime*, more flexibility is achieved, since each environment can define its own local policies to rule interactions.

3.1. The Role level. In our proposal, the upper level of an infrastructure is composed by a set of roles related to the same application context. Such definition implies two important features of an infrastructure:

- it is not bound to specific agents, which can belong to whatever applications, can have their own tasks and can be designed and implemented separately from the environment;
- it can host agents, providing a "wrapper" that not only accepts them, but also assigns them capabilities and a specific behaviour.

In Figure 3.1 the set of roles is represented by a "table" where each role is represented by a "hole", in which an agent can place itself in order to play such role. There could be several "holes" of the same role, if an environment can host more than one agent playing the same role.

This idea of infrastructure enforces the locality concept previously introduced. In fact, each environment can decide how to organize the local hosting of agents and, by defining also mechanisms and policies, how local interactions are ruled.

3.2. The Policy & Mechanism Level. This level deploys the policies that rule the local environment, and provides the mechanisms for the interactions both between agents and between agents and the resources of the environment. While the previous level can be considered as the interface of an environment toward the external world, this level enacts the environment's laws. The simplest example of policy is to allow or deny an interaction between two given roles. Thanks to their autonomy and reactivity, agents can handle situations where something is forbidden by local rules without giving up and aborting their job.

Even if different from policies, we include mechanisms at this level because, as policies, they enable the interactions between roles and with local resources. To be more precise, this level should be split into two sub-levels: a policy one and a mechanism one, since policies rely on mechanism; but from our point of view this distinction is not relevant.

3.3. The Resource Level. At the lowest level we can find the resources local to an environment. They can be legacy resources that are hard to change or affect. So, it is important that the policy & mechanism level makes them available in a useful format for agents. As mentioned, this level is considered as part of the infrastructure, but modelling resources as outside the infrastructures does not mine the proposed model.

Also in this case, the use of roles helps in abstracting from the single agent or application, because mechanisms has to be enacted for a generic role, covering the wide range of actual agents that play such role.

Our proposal permits to disregard how local resources are managed, providing that appropriate access mechanisms are supplied.

4. Implementing Role-based Infrastructures. The infrastructure abstraction must have its concrete counterpart, i. e., an appropriate implementation. We are currently exploring the implementation of the proposed role-based infrastructures exploiting RoleX [3], which is a role-based interaction system implemented in the context of the BRAIN framework [10]. RoleX considers roles as first-class entities and enables agents to dynamically assume, use and release roles; the idea behind its implementation comes from the Aspect Oriented Programming [11], even if it has been re-adapted to the agent scenario. Moreover, RoleX is inspired by real life, thus provides a high degree of freedom to agents that want to assume and play roles. The real life inspiration of RoleX is emphasized by the role *external visibility* that it provides: the role an agent is playing can be identified directly from an external point of view (e.g., another agent) without requiring to ask the interested agent for it. RoleX is a pure Java middleware and is not another agent platform. RoleX can be easily associated with any Java agent platform. RoleX adopts the BRAIN XML notation to describe roles. The adoption of XML provides several advantages, as described later.

We have chosen RoleX for three main reasons.

First, being part of the BRAIN framework, it supports the different phases of the software development, from design to implementation [4], and we think this is a valid help to developers, which can have coherent development phases and not fragmented solutions. Moreover, RoleX supports also dynamic assumption of roles at runtime, granting a high degree of flexibility.

Second, we aim at exploiting the XML notation to produce definitions of roles that are interoperable, flexible, and expressive. Each role can be defined by means of an XML document compliant to a given XML Schema, which is independent of the language of its implementation; then XML allows developers to express information in a tagged and structured way, both human- and machine-readable; finally, XML documents can be translated into other formats (e. g., HTML) in order to be catalogued or viewed in a more suitable way.

Third, RoleX provides some mechanisms to define interaction rules, and this turns out useful to define the policies of the infrastructures. Among such mechanisms, we can mention some flexible security mechanisms [5], which are based on the Java Authentication and Authorization System (JAAS) architecture [28], and enable a fine-grain control over the allowed operations of roles. Then, a simple yet useful mechanism permits to define which roles can interact with which other roles; this is useful, for instance, to avoid collusion in an auction context. Finally, perhaps trivial, an agent can assume a role only if it has the right permissions.

RoleX has been exploited also to implement *computational institutions* [6], where it has proved to be a powerful and flexible means to implement abstractions.

4.1. Role Description and Implementation. To grant a high level of abstraction in deciding which role to play, and to cope with dynamic situations, it is important to uncouple roles from their implementation; this allows agents to focus on the semantics of the roles, rather than their code. To this purpose RoleX uses *role descriptors*, which are entities that describe a role, for example by means of information such as keywords, a contest, an aim, a version, a creation date and any further needed piece of information. Descriptors are defined by XML documents written exploiting notation of BRAIN. Besides relieving programmers of the knowledge of role implementation, the descriptors are useful also to hide to the agent the physical location of the role implementation, to enable role composition, to change role implementation in a transparent way, and to allow the *agent* programmers to disregard about the work of *role* programmers and viceversa.

Since the agents are developed using Java, our implementation enables automatically the translation from the XML documents representing descriptors to a set of Java classes. In this way, an agent can directly access the descriptors without needing an XML-parser. The Java implementation of a role is composed of two parts: a Java interface (*role interface*) and a Java class (*role implementation*); the interface provides external visibility (simulating multiple inheritance) while the class provides the effective role behavior.

A role assumption means that the RoleX system performs run-time bytecode manipulation, and in particular (i) adds each role class member (both methods and fields) to the agent class, in order to add the set of capabilities of the role and, at the same time, (ii) forces the agent class to implement the role interface, in order to modify its appearance and to allow other agents to recognize it as playing that role.

Since the above mechanism must result in the definition of a new class, our approach exploits a special class loader, called `RoleLoader`, that can change the agent behavior and the external appearance. It is a subclass of SecureClassLoader, which allows us to work in compliance with the Java Security Manager. After the `RoleLoader` has successfully carried out the role assumption process (i. e., the addition of the members and the interface), it can reload the agent restarting it. The assumption process can be briefly described by Figure 4.1, where the agent searches a role repository for an appropriate role descriptor, for example by using keywords, and then asks the `RoleLoader` to reload itself with the chosen role among the retrieved ones. The `RoleLoader` retrieves the implementation corresponding to the role descriptor and adds it to the agent. If everything goes right, the `RoleLoader` sends the new agent an event (called reload event) to indicate that the agent has been reloaded. After the reload event, the agent can resume its execution. Releasing a role is similar to the above process, but this time the `RoleLoader` removes each role member and the role interface, reloading the agent without them.

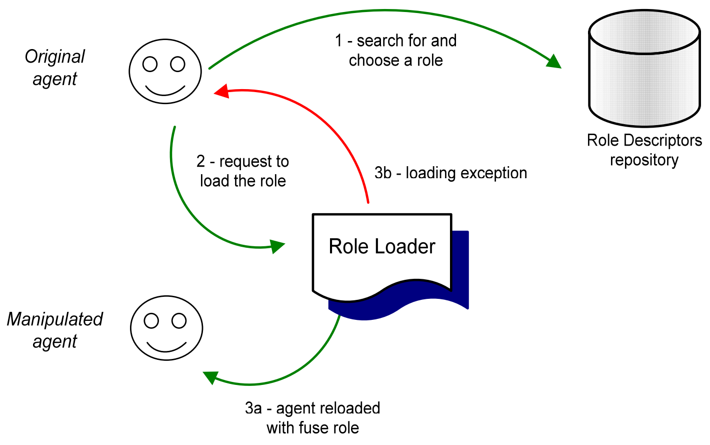
Original agent
1 - search for and choose a role
Role Descriptors repository
2 - request to load the role
3b - loading exception
Role Loader
Manipulated agent
3a - agent reloaded with fuse role

Fig. 4.1. *Role assumption process*

The exploitation of RoleX in the definition of role-based infrastructures will be explained in the next section, by means of a concrete example.

5. Application Examples. This section presents a couple of examples of applications where a role-based infrastructure is defined with the corresponding policies and mechanisms.

5.1. Auctions. The first example relates to auctions. Auctions represent an interesting negotiation means in the agent area, and we have already exploited them in this paper to support some concepts. In an auction there are entities (called *sellers*) that make goods/resources available and entities (called *bidders*) that are interested in using/acquiring such goods/resources. Moreover, there are intermediate entities (called *auctioneers*) in charge of actually performing the negotiation. The price of the resources sold by sellers via an auction is not fixed, but it is dynamically determined by the interest of the bidders [1].

We can figure out that agents negotiate resources or goods via auctions, at given sites representing auction houses [25]. Of course, the way the sellers, the bidders and the auctioneers interact is not bound to a given application or to a given environment, and so they can be considered roles that whatever agent can assume. In this case, the choice of the roles is tightly driven by the kind of application: the *bidder*, the *seller* and the *auctioneer* (see Figure 5.1). So the former step of Section 3 is accomplished.

Figure 5.2 and 5.3 report possible XML descriptions of the bidder and the seller role respectively. The site is in charge of providing role implementations, which are likely to depend on the local policies or mechanisms. From the agent point of view, these descriptions can be exploited to search for an appropriate role in a site and, once found, it can assume the role, placing itself in the “hole” of the site. From the site point of view, the roles made available represent the interface by which the environment can host agents; moreover, roles can be implemented and managed meeting local requirements and in compliance with local polices. In fact, the latter step relates to the choice of the local policies of interaction between roles and between roles and the environment resources.

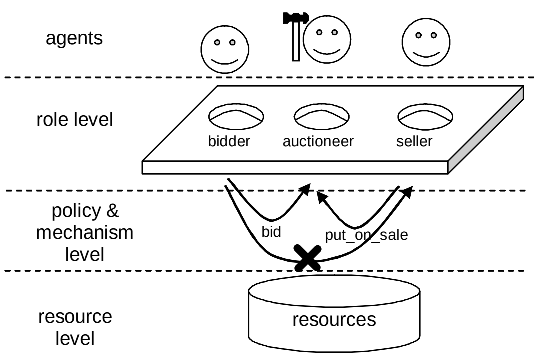

Fig. 5.1. *A role-based infrastructure for an auction house*

```
<?xml version="1.0" encoding="UTF-8"?>
<role xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="XRole.xsd">
 <name>bidder</name>
 <context>auction</context>
 <description>
 This role is the bidder of an auction.
 </description>
 <keyword>auction</keyword>
 <keyword>bidder</keyword>
 ...
 <action>
 <name>bid</name>
 <description>Makes a bid.</description>
 <content>
 <description>Bid.</description>
 <type>Price</type>
 </content>
 </action>
 ...
 </role>
```

Fig. 5.2. *A bidder role defined in XML*

There can be several reasons to have different policies or mechanisms. For example, in one environment the bidders could be allowed to talk each other, while in another environment they cannot, to avoid collusions; this permits to impose local rules or social conventions [32]. For instance, Figure 5.4 reports the possible allowed interactions expressed by a grid whose definition is enabled by RoleX. As in Figure 5.1, bidders and sellers are not allowed to interact directly.

Another reason could be the different implementation of the auction mechanisms: Figure 5.1 shows a message-passing oriented implementation, where, for instance, the bidder agent can bid by sending a message to the auctioneer agent. But if the implementation of the bidding mechanism is based on another model, the local policies must be different. For instance, if the auction relies on a data-oriented model such as tuple spaces [7], the bidding action is implemented as writing information in the local interaction space, as shown in Figure 5.5. This example shows that the same set of roles can be adapted to different implementations.

5.2. Restaurants. This example is taken from the human life, and may not be related to a real application based on agents; however, it is meaningful to understand how roles can be defined and how the interactions among them can be established.

In this example, a node represents a single restaurant. We define three roles: the *customer*, the *waiter*,

```
<?xml version="1.0" encoding="UTF-8"?>
<role xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="XRole.xsd">
 <name>seller</name>
 <context>auction</context>
 <description>
 This role is the seller of an auction.
 </description>
 <keyword>auction</keyword>
 <keyword>seller</keyword>
 ...
 <action>
 <name>put_on_sale</name>
 <description>Put a good on sale.</description>
 <content>
 <description>Good.</description>
 <type>String</type>
 </content>
 </action>
 ...
 </role>
```

Fig. 5.3. *A seller role defined in XML*

	Bidder	Seller	Auctioneer
Bidder	No	No	Yes
Seller	No	No	Yes
Auctioneer	Yes	Yes	Yes

Fig. 5.4. *Allowed interactions*

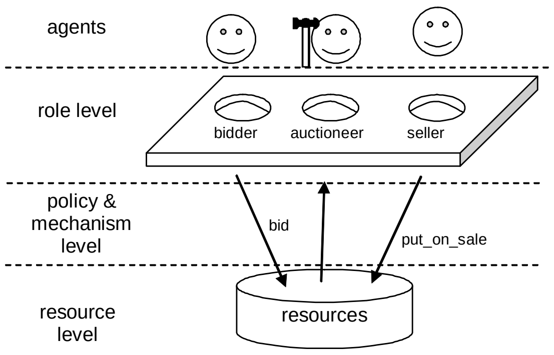

Fig. 5.5. *The same infrastructure relying on a data-oriented model*

and the *chef* (see Figure 5.6). These roles can be thought as instances of the more general roles defined in a client-server model with an intermediate entity (the waiter) between the client (the customer) and the server (the chef), such as several 3-tier solutions.

The role of the customer can have the following capabilities: *ask for the menu*, *order the meal*, *accept the meal*, and *pay the bill*. Note that "*eat the meal*" is not a capability of the role of customer, while it should be of the agent. The waiter role has different capabilities: *take order*, *order the meal* (to the chef), *accept the meal* (from the chef), *give the meal* (to the customer), and *accept the payment*.

Finally, the chef can: *accept an order* and *give the meal*. Again, the cooking of the meal is not an external capability of the chef *role*, but an intrinsic capacity of the chef *agent*.

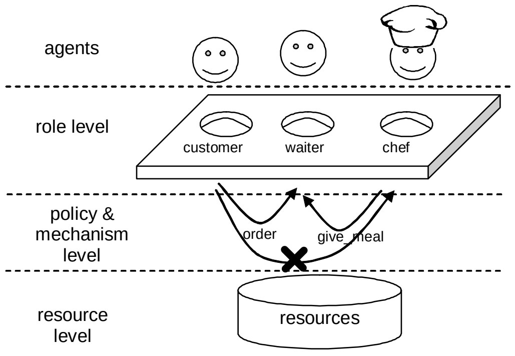

Fig. 5.6. *A role-based infrastructure for a restaurant*

The policy & mechanism level ensure that such interactions occur, for example it grants that the customer orders the meal to the waiter and the chef gives the cooked meal to the waiter. Some interactions may be disabled, such as the direct interaction between the customer and the chef (see Figure 5.6).

Now, let us suppose that the scenario changes. To save money, little restaurants do not have the waiter, but the chef itself is in charge of accepting and satisfying the customers' requests (see Figure 5.7).

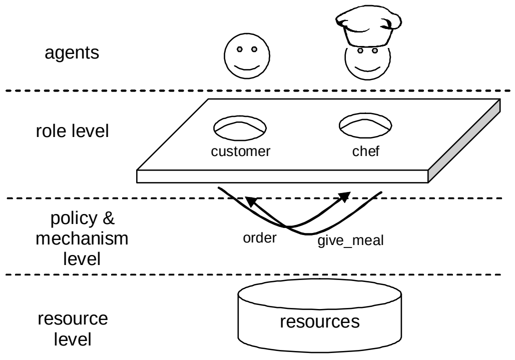

Fig. 5.7. *A restaurant without waiters*

In this case, the infrastructure can exploit the same roles - disregarding of course the waiter role. On the one hand, the agents can assume the same roles as in the previous scenario, and they can perform the same actions in the restaurant. On the other hand, the policy and mechanism level must be changed in order to allow the new interactions, letting the customer give the meal order to the chef, and the chef give the meal to the customer.

This example shows that the same roles can adopt different interaction patterns without affecting the role definitions and, as a consequence, the agents' way to achieve their goals.

The implementation of this example is similar to the previous one.

6. Related Work. In this section we report some role-based approaches for agents.

Elizabeth Kendall has done a lot of work about roles and agents [21], and her proposal is perhaps the most complete. Her effort aims at covering different phases of the agent-based application development, from the analysis to the implementation. To achieve this, her proposal exploits both Object Oriented and Aspect Oriented Programming (AOP) [11] in order to make flexible, reusable and dynamic the use of roles. With regard to role-based infrastructures, an interesting proposal is represented by *role model catalogues*, which are collections of role models used in agent applications. Even if not thought for infrastructures, they can be exploited to group roles belonging to the same application context.

AALAADIN [15] is a role approach that exploits a meta-model to define models of organizations. Such meta-model is based on three main concepts: *agent*, *group* and *role*. The concept of group can be exploited to model the infrastructures proposed in this paper. A drawback of this approach is that it covers only the analysis and design phase, while no support for the implementation phase is provided, which can turn useful to have coherence in the entire development process.

Yu and Schmid [31] exploit roles assigned to agents to manage workflow processes. They traditionally model a role as a collection of rights (activities an agent is permitted on a set of resources) and duties (activities an agent must perform). An interesting issue of this approach 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, that is partially supported. Joining this approach with some means to define infrastructures can lead to a complete and interesting proposal.

The Tractable Role-based Agent prototype for concurrent Navigation Systems (TRANS) is a multi-agent system with support for role and group behaviours [16]; this approach explicitly takes into consideration that agents can be mobile. Moreover, TRANS allows the definition of rules on the role assumption by agents, such as priority, exclusivity, compatibility and the distinction between permanent and temporary roles. Even if designed for a specific application scenario, TRANS provides an interesting role model and, in particular, a support for group of roles. A possible drawback is that the implementation lacks of dynamism, and thus it is not appropriate for agent applications for dynamic and unpredictable scenarios.

Another interesting role approach is GAIA [30]. The GAIA main aim is modelling agent systems as *organizations* of agents, where different roles interact. In GAIA, roles are considered only in the analysis phase, and a role definition includes concepts like responsibilities, permissions, activities and interaction protocols. GAIA focuses on role interactions and supports a model to describe dependencies and relationships between different roles. The above model is supported at the design phase and is made of a set of *protocol definitions*, each of them related to the kind of interaction among roles. Similarly to the above approaches, even GAIA lacks on the implementation support and provides a quite strict definition of role related concepts, leaving freedom to developers.

Maria Fasli made a proposal that joins several concepts, such as commitments and obligations, with the powerful of roles, promoting the use of a formal notation and analysis of the applications [14]. The base idea of this proposal is that multi-agent systems are composed of *social agents*, which are social since they do not act isolated. The term *social agent* refers either to a single agent or to 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. 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 accordingly to it. Even if this proposal offers a comprehensive view of social and collective activities, describing them as social and collective concepts, there is no concrete support at the implementation level.

The Role/Interaction/Communicative Action (RICA) theory [26] was born with the main aim of improving the FIPA standard with support for social concepts. The RICA theory fuses communication concepts with social ones, and in particular merges FIPA-ACL and organizational models, keeping all concepts as first class entities. The RICA theory recognizes *communicative entities*, which are those that can be aggregated and organized in a social way; in other words communicative entities are agents acting in a society. Each agent (type) is defined through the role (types) it will play; each role can perform one or more action, implementing a social behaviour. Action can be specialized in social and communicative ones, which emphasizes the above statement. Finally, each role can be specialized as a social role, which represents the behaviour of agents interacting in a social context. The RICA metamodel can be used as a formal language, providing support for the analysis and design phase of an agent application. Thanks to the RICA-J (RICA Jade) implementation, this proposal is complete and can be used during all phases of application development. Probably the most important drawback of this proposal is that it re-defines the concept of agent, leading to a possible incoherence with other agent theories and approaches.

In [33] Haibin Zhu describes a role model that is tied to both the computer and human parts involved in collaborations, and in particular tries to provide help to human in computer-supported collaborations. This model defines a role as a set of *responsibilities* and *capabilities* a human holds. The key of this proposal is the concept of *role agent*, which is an agent with a role attached. An agent that wants to participate in a 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 run-time, 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. This proposal exploits a formal notation, 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. Most of the concepts of this proposal, such as groups and messages, are not new, and the definition of roles as set of messages is not really flexible. Nevertheless this proposal well reflect a real situation, and being based on a formal notation, can help designers and developers planning the system.

7. Conclusions and Future Work. Multiagent systems must be carefully designed and developed. This paper has proposed an approach to agent composition based on roles, which are exploited to build infrastructures for agents. This permits to face the development at different levels, ruling both capabilities and restrictions, and enabling the implementation of policies and mechanisms depending on the local context.

The adoption of the RoleX system has had a twofold advantage. On the one hand, it has enabled the concrete implementation of the approach; the proposed example has shown how the different levels of the role-based infrastructures can be implemented or supported. On the other hand, RoleX has exhibited a great degree of flexibility, allowing the uncoupling between role descriptions and role implementations.

With regard to future work, we are exploiting the RoleX system to implement role-based infrastructures for agents, as already mentioned. RoleX enables the dynamic assumption of roles by agents, letting environments define role repositories with their own implementation of roles. We are going to study on the one hand the applicability of the proposed role-based infrastructures, and on the other hand the flexibility of the RoleX system.

Then, effective tools in the field of software engineering are to be developed to support the building of infrastructures. They can help both the environment developers and also the environment administrators, which can decide to change the local policies or mechanisms.

Acknowledgments. This work was supported by the European Community within the project CASCADAS.

The author thanks the anonymous authors for their precious comments.

REFERENCES

[1] Agorics, Inc., *Going, going, gone! A survey of auction types*, `http://www.agorics.com/new.html` 1996.

[2] D. Baumer, D. Ritchie, W. Siberski, and M. Wulf, *The Role Object Pattern*, Proceedings of the 4th Pattern Languages of Programming conference (PLoP), Monticello, Illinois, USA, September 1997.

[3] G. Cabri, L. Ferrari, and L. Leonardi, *The RoleX Environment for Multi-Agent Cooperation*, The 8th International Workshop on Cooperative Information Agents (CIA), Erfurt, Germany, Lecture Notes in Artificial Intelligence No. 3191, Springer-Verlag, September 2004.

[4] ———, *Supporting the Development of Multi-Agent Interactions via Roles*, the 6th International Workshop on Agent-Oriented Software Engineering (AOSE) at AAMAS 2005, Utrecht, The Netherlands, July 2005.

[5] ———, *Applying Security Policies Through Agent Roles: a JAAS Based Approach*, Science of Computer Programming, (Elsevier, Amsterdam-NL), Vol. 59, No. 1-2, January 2006, pp. 127–146.

[6] G. Cabri, L. Ferrari, and R. Rubino, *Building Computational Institutions for Agents with RoleX*, technical report MO/AG/05/001.

[7] G. Cabri, L. Leonardi, and F. Zambonelli, *Auction-based Agent Negotiation via Programmable Tuple Spaces*, Proceedings of the 4th International Workshop on Cooperative Information Agents (CIA 2000), LNCS No. 1860, Boston (USA), July 2000.

[8] ———, *Engineering Mobile Agent Applications via Context-dependent Coordination*, IEEE Transactions on Software Engineering, Vol. 28, No. 11, November 2002, pp. 1040–1056.

[9] ———, *Modeling Role-based Interactions for Agents*, The Workshop on Agent-oriented methodologies at the 17th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2002), Seattle, Washington, USA, November 2002.

[10] ———, *BRAIN: a Framework for Flexible Role-based Interactions in Multiagent Systems*, The 2003 Conference on Cooperative Information Systems (CoopIS), Catania, Italy, November 2003.

[11] *Communication of the ACM, Special Issue on Aspect Oriented Programming*, Vol. 33, No. 10, October 2001.

[12] S. Clearwater, *Market-based Control: a Paradigm for Distributed Resource Allocation*, World Scientific, 1995.

[13] B. Demsky and M. Rinard, *Role-Based Exploration of Object-Oriented Programs*, Proceedings of the International Conference on Software Engineering 2002, Orlando, Florida, USA, May 19-25, 2002.

[14] M. Fasli, *Social Interactions in Multi-Agent Systems: A Formal Approach*, The First European Workshop on Multi-Agent Systems (EUMAS), 18-19 December 2003, Oxford, UK

[15] J. Ferber and O. Gutknecht, *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), 1998.

[16] S. Fournier, D.Brocarei, T.Devogele, and C. Claramunt, *TRANS : A Tractable Role-based Agent Prototype for Concurrent Navigation Systems* , The First European Workshop on Multi-Agent Systems (EUMAS), 18-19 December 2003, Oxford, UK.

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

[18] N. R. Jennings, *An agent-based approach for building complex software systems*, Comm. of the ACM, Vol. 44, No. 4, 2001, pp. 35–41.

[19] N. R. Jennings, M. Wooldridge, eds., *Agent Technology: Foundations, Applications, and Markets*, Springer-Verlag, March 1998.

[20] N. M. Karnik and A. R. Tripathi, *Design Issues in Mobile-Agent Programming Systems*, IEEE Concurrency, Vol. 6, No. 3, July-September 1998, pp. 52–61.

[21] E. A. Kendall, *Role Modelling for Agent Systems Analysis, Design and Implementation*, IEEE Concurrency, Vol. 8, No. 2, April-June 2000, pp. 34-41.

[22] B. B. Kristensen and K, Øterbye, *Roles: Conceptual Abstraction Theory & Practical Language Issues*, Special Issue of Theory and Practice of Object Systems on Subjectivity in Object-Oriented Systems, Vol. 2, No. 3, 1996, pp. 143–160.

[23] *Compact Oxford English Dictionary online*, `http://www.askoxford.com`

[24] R. S. Sandhu, E. J. Coyne, H. L. FeinStein, and C. E. Youman, *Role-based Access Control Models*, IEEE Computer, Vol. 20, No. 2, 1996, pp. 38-47.

[25] T. Sandholm and Q. Huai, *Nomad: Mobile Agent System for an Internet-Based Auction House*, IEEE Internet Computing, Special issue on Agent Technology and the Internet, Vol. 4, No. 2, March-April 2000, pp. 80–86.

[26] J. Manuel Serrano and S. Ossowski, *On the Impact of Agent Communicative Languages on the Implementation of Agent Systems*, Cooperative Information Agents VIII, M.Klush, S. Ossowski, V. Kashyap, R. Unland eds., Lecture Notes in Artificial Intelligence, ISSN 0302-9743, Springer, 2004

[27] B. Shneiderman and C. Plaisant, *The Future of Graphic User Interfaces: Personal Role Manager*, Proceedings of the conference on People and computers IX, Glasgow 1994

[28] SUN Microsystems, *Java Authentication and Authorization Service (JAAS)*, `http://java.sun.com/products/jaas/`

[29] A. Tripathi, T. Ahmed, R. Kumar, and S. Jaman, *Design of a Policy-Driven Middleware For Secure Distributed Collaboration*, Proceedings of the 22nd International Conference on Distributed Computing System (ICDCS), Vienna (A), July 2002.

[30] M. Wooldridge, N. R. Jennings, and D. Kinny, *The Gaia Methodology for Agent-Oriented Analysis and Design*, Journal of Autonomous Agents and Multi-Agent Systems, Vol. 3, No. 3, 2000, pp. 285–312.

[31] L. Yu, and B. F. Schmid, *A conceptual framework for agent-oriented and role-based workflow modelling*, in Proceedings of the 1st International Workshop on Agent-Oriented Information Systems, G. Wagner and E. Yu eds., Heidelberg, June 1999.

[32] F. Zambonelli, N. R. Jennings, and M. Wooldridge, *Organizational Rules as an Abstraction for the Analysis and Design of Multi-agent Systems*, Journal of Software Engineering and Knowledge Engineering, 2001.

[33] H. Zhu, *A Role Agent Model for Collaborative Systems*, Proceedings of the 2003 International Conference on Information and Knowledge Engineering (IKE'03), Las Vegas, Nevada, USA, June, 2003, pp. 438–444.

Edited by: Shahram Rahimi, Raheel Ahmad
Received: December 9, 2005
Accepted: March 19, 2006