Submission to The International Workshop on AGENT-ORIENTED SOFTWARE ENGINEERING (AOSE-2005)

Supporting the Development of Multi-Agent Interactions via Roles

Giacomo Cabri, Luca Ferrari, Letizia Leonardi

Dipartimento di Ingegneria dell'Informazione - Università di Modena e Reggio Emilia
Via Vignolese, 905 – 41100 Modena – ITALY
{cabri.giacomo, ferrari.luca, leonardi.letizia}@unimo.it

CONTACT AUTHOR:
Giacomo Cabri
Dipartimento di Ingegneria dell'Informazione
Università di Modena e Reggio Emilia
Via Vignolese, 905 – 41100 Modena – ITALY
Tel. +39-059-2056190
Fax +39-059-2056126
Email cabri.giacomo@unimo.it

Supporting the Development of Multi-Agent Interactions via Roles

Giacomo Cabri, Luca Ferrari, Letizia Leonardi

Dipartimento di Ingegneria dell'Informazione - Università di Modena e Reggio Emilia
Via Vignolese, 905 – 41100 Modena – ITALY
{cabri.giacomo, ferrari.luca, leonardi.letizia}@unimo.it

Abstract. In the multi-agent scenario, interactions among agents are an issue that must be faced in an appropriate way. Modeling interactions by roles can simplify the development of the interactions in agent-based applications. The BRAIN framework proposes an interaction model based on roles, an XML notation to define roles, and interaction infrastructures based on the role model and notation. In this paper we explain how the BRAIN framework can be exploited in the different phases of the development of applications where agents play roles. The general advantage is that the development phases rely on the same information, adapted to different needs, granting coherence and continuity during the development.

1 Introduction

Multi-agent systems are gaining more and more ground in the development of complex applications [Jen01]. The advent of agents raised the problem of dealing with interactions either in a competition or collaboration context. In fact, one of the main features of agents is sociality, i.e., the capability of interacting each other. This feature is exploited in Multi Agent Systems, where the main task is divided into smaller tasks, each one delegated to a single agent; agents belonging to the same application have to interact in order to carry out the main task [FalHM99]. Moreover, the diffusion of open systems, such as the Internet, has led to a scenario in which not only agents of the same application interact in a cooperative way, but also agents of different applications may interact in a competitive way, for example to use resources. The feature of mobility, which allows agents to change their execution environment, adds great flexibility at both conceptual and implementation levels, but also introduces peculiar issues in interaction that must be taken into account carefully [DomLD97].

Interactions are not an exclusive field of agent technology, and in fact, since the rising of distributed systems, the interaction among entities has been an issue to be faced. Different approaches have been proposed, among which the most popular is message-passing [Cha90, Wal94]; it enables the exchange of messages between two entities, and it is simple and suits a wide range of application requirements. Another interaction approach that is worth being mentioned is Linda [AhuCG86], which is based on an uncoupled coordination model and relies on shared dataspaces.

The above general approaches have been ported to the agent scenario [CabLZ00a], creating the “meeting point” abstractions [Whi97], event-channels [Bau98], and reactive tuple spaces [CabLZ00c]. Further approaches have been proposed, like the one adopted in Agentis [InvKL98], which models interactions in terms of services and tasks; such approach clearly points out the need of tailoring interactions on agent features.

In fact, approaches to agent interactions usually derive from adaptations of traditional models related to distributed systems, so they do not fit all the characteristics of the new scenario. Moreover, it is important to provide to developers appropriate tools to represent interactions and the way they happens, thus it is likely that a notation based on the UML language will have a more widespread success than other proposals [OdePP01, Lin01].

Recognizing the limitations of the traditional approaches, we have proposed BRAIN, a framework to deal with agent interactions based on the concept of *role*. There are different advantages in modeling interaction by roles. First, it enables the separation of concerns between the algorithmic issues and the interaction issues in developing agent-based applications. Second, it permits the reuse of solutions and experiences; in fact, roles are related to a context, and designers can exploit previously defined roles for the context their application belongs; therefore roles can be seen as sort of design patterns [AriLan98]: a set of related roles along with the definition of the way they interact can be considered as a solution to a well-defined problem, and reused in different similar situations. Finally, roles can be exploited to easily build agent-oriented interfaces of Internet sites [CabLZ03b].

In this paper we explain how the BRAIN framework can be exploited during the development of applications based on roles and agents, and in particular we report the advantages gained during the four phases: analysis, design, implementation and runtime execution.

The paper is organized as follows. Section 2 presents the BRAIN framework. Section 3 explains in detail how the BRAIN framework can be exploited during the phases of the development process. Section 4 shows a simple example of application. Section 5 reports some related work. Finally, Section 6 concludes the paper and sketches some future work.

2 BRAIN

The BRAIN (Behavioural Roles for Agent INteractions) framework [BRAIN02] is based on the concept of *role* and aims at covering the agent-based application development at different phases. To this purpose, it provides for a *model of interactions* that is based on roles, an XML-based *notation* to describe the roles, and *interaction infrastructures* supporting agents in the management of roles. Such infrastructures are based on the adopted model and rely on the defined XML notation (see Figure 1).

Each part of the BRAIN framework will be detailed later, but what is important to note here is that, since this framework is made by three layered levels, it can be easily adapted to different application scenarios and can evolve keeping a good modularity.

For example, as shown in Figure 1, BRAIN can support several interaction infrastructures, running under the same XML notation and role model. Similarly, the XML notation can change, due to the exploitation of a different XML-Schema, without requiring changes to the role model or to the current interaction infrastructure. Moreover, thanks to its layered organization, the BRAIN framework is able to support the development of applications at all the main phases (i.e., analysis, design, implementation) together with the runtime execution.

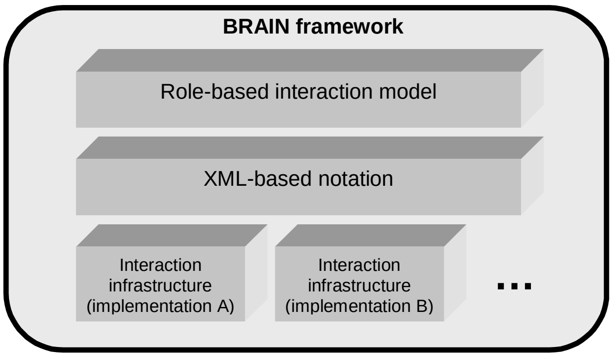
BRAIN framework
Role-based interaction model
XML-based notation
Interaction infrastructure (implementation A)
Interaction infrastructure (implementation B)
...

Figure 1. The BRAIN framework

2.1 Role Model

In BRAIN, a role is defined as a *set of capabilities* and an *expected behavior*. The former is a set of *actions* that an agent playing such role can perform to achieve its task. The latter is a set of *events* that an agent is expected to manage in order to "behave" as requested by the role it plays. Interactions among agents are then represented by couples (action, event), which are dealt with by the underlying interaction system, which is part of the BRAIN infrastructure; the interaction system can control interactions and enforce local policies, such as allowing or denying interactions between agents playing given roles. Note that all policies applied to roles must be considered as additions to the programming language policy management, such as the Java security manager. This means that all policies applied into the BRAIN infrastructure can be applied in a separate way from the Java policy system, which can be kept enabled. Figure 2 shows how an interaction between two agents occurs. This model of interactions is very simple and very general, and well suits the main features of the agents: the actions can be seen as the concrete representation of agent *proactiveness* (i.e., the capability of carrying out their goals), while the events reify the agent *reactivity* (i.e., the capability of reacting to environment changes).

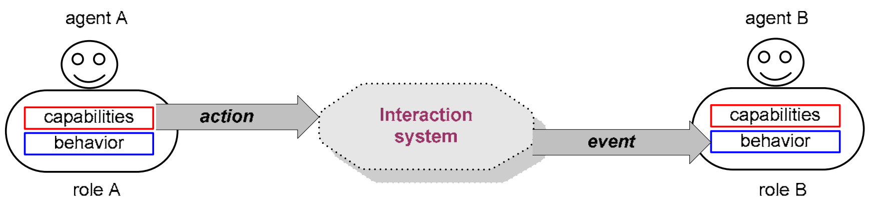

Figure 2. The Interaction model in BRAIN

2.2 XRole

Starting from the role model described in the previous section, we have developed XRole [CabLZ02a], a notation for the definition of roles for agents. The main aim of XRole is to support the management of the interactions among agents, at the different phases of the development of agent-based applications. XRole adopts the model of role of BRAIN, based on *actions* and *events* (see Figure 2).

XRole exploits the XML language to build up definitions of roles. XML, representing information in a tagged form, exhibits the well-appreciated feature of human-readability and the platform-independence required for the Internet. This allows a high degree of interoperability and helps in dealing heterogeneous situations such as the ones of the Internet. By using XML, the description of each role can be presented to people via an appropriate XSL sheet that transforms the information in a human-understandable document, for instance an HTML page with the needed pieces of information. This makes designers aware about the roles defined in a given context, and let them choose the representation of the information that better suits their requirements. In addition, programmers can exploit automatic tools to search for roles matching given keywords and which present the found results in a tailored way. A further advantage of XML is that documents can be managed also by the agents themselves, which can be enabled to understand the content of XML documents and exploit or manipulate them without the need of the intervention of human people.

XML documents can be used to construct role catalogues, that are collections of documents describing available roles, providing support for the choice of the right role to agents and their developers. Exploiting XML to build role catalogues, grants advantages, such as the capability to use the same document to both describe the role and the operative use of it. Moreover, the advantage of exploiting the XML language is that this definition of role can be extended, to meet specific requirements that will arise in the future.

In XRole, roles are defined by XML documents that respect the XML Schema reported in Figure 3.

The XML Schema for roles shows that there are three main parts that have to be specified in the definition of a role, following the above-described model:

- *The basic information*. This part includes the pieces of information that are used to identify the role, and to specify a context for such role. Moreover, a high-level description and some keywords are supplied to let developers better understand the role functionalities.

- *The allowed actions*. Each role defines which are the allowed actions, i.e. the actions needed to carry out a task related to such role. An action is characterized by a name and a high-level description that can be useful for developers. Moreover, there are two elements that specify the addressee of the action and the content, which can be exploited for further information.
- *The recognized events*. These are events that the agent is expected to accept and manage. An event, as an action, is characterized by a name and a high-level description that is useful to developers. Other two elements specify who performed the related action and the event content, which again is very general.

Figure 3 shows an example of an XRole document for a game player role (see section 4 for further details), a role assumed by an agent that wants to participate to a gaming session. The description of the role, included between the <GenericRoleDescription> tags, includes a few information about the role (basic information) such as a description, the role name, the keywords, the version number, etc. Following that, it is possible to note the description of an action (tag <OperationDescription>), which in particular is an action/operation that allows the agent to ask to the game manager what kind of game is currently running. This operation will produce an incoming event, described in the <EventDescriptor> tags, that the agent should handle, since it represents the reply to the above question.

```
<?xml version='1.0'?>
<role xmlns="http://agentgroup.unimo.it/schema/BRAINSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:SchemaLocation="http://agentgroup.unimo.it/schema/BRAINSchema" >


 <GenericRoleDescription>
   <description>Game Player Role</description>
   <roleName>player</roleName>

   <!-- data useful to identify this role -->
   <keyword>game</keyword>
   <keyword>player</keyword>
   <keyword>make moves</keyword>
   <version>1</version>

   <!--  an operation of this role -->
   <OperationDescription>
     <name>askKind</name>
     <aim>get the kind of current game</aim>
     <keyword>kind</keyword>
     <keyword>request</keyword>
      ...
      <EventDescriptor>
         <name>notifyKind</name>
         <aim>Informs which kind of game is currently on</aim>
         . . .
         <ReceivingEvent>true</ReceivingEvent>
      </EventDescriptor>

    </OperationDescription>
   <!--  other role actions/operations here -->
   ...

   <!--  an event that can be received even without an action -->
   <EventDescriptor>
      <name>requestMove</name>
      <aim>Request the player to make a move</aim>
      . . .
      <ReceivingEvent>true</ReceivingEvent>
   </EventDescriptor>

 </GenericRoleDescription>
```

Figure 3. An example of a XRole document for the game player role.

It is important to note, as already emphasized, that thanks to the XRole notation, a developer can easily understand what a role can do, and so how its assumption can be appropriate for the application context the agent is living in. Moreover, having a look at the <EventDescriptor> tags, it is possible to immediately understand what consequences the execution of a role action/operation will have on the surrounding environment and/or on other agents. For example, with regard to the example shown in Figure 3, when an agent performs the askKind action, an other agent will reply to such action with an event (notifyKind), that is delivered to the agent that has performed the askKind action.

It is important to note that, as shown in Figure 3, one or more events are not strictly tied to a role action, that means an event can be delivered even if none role action has been executed yet. This is the case, for example, of the requestMove event, delivered to the player agent to inform that it must make a move to continue the game; there is not a correspondent previous action in the player role that accepts this event as a reply.

2.3 Interaction Infrastructures

BRAIN can adopt several interaction infrastructures, but they must rely on the XRole notation and must support the role model defined on the top of the BRAIN framework. Being able to change the interaction infrastructure allows BRAIN to adapt the role model depending on runtime constraints or parameters, choosing the most appropriate implementation for a specific scenario. Of course, the adoption of a interaction infrastructure instead of another does not affect the role model, that is how agents (and their roles) interacts, but only how these interactions are implemented and happen at runtime.

Currently, there are two interaction infrastructures compliant with BRAIN: RoleSystem [CabLZ03b] and RoleX [CabFL04a]. Due to space limitation, this paper will not cover in detail the above infrastructures, but just to understand the main difference between them, it suffice to say that RoleX exploits some Java techniques to realize a life-inspired role system implementation [CabFL04a], resulting in a more heavy environment to run, while RoleSystem requires less resources and thus is more appropriate to be run on the top of small devices (PDAs, mobile phones, etc.). More details can be found in [CabLZ03b].

3 Development Phases Through BRAIN

In this section, we present the advantages that the BRAIN framework, and the XRole notation in particular, can provide in the three main phases of software development: analysis, design, implementation. Furthermore, this section will illustrate also how BRAIN supports the runtime execution that, even if not strictly tied to the application development (excluding the debugging), is important from a software life-cycle point of view.

As readers will see, an important feature of BRAIN is to exploit the XRole notation in the middle of the framework, that allows the framework itself to exploit the same well formatted information along different phases. This leads to managing always the same information, even if represented in different ways depending on the specific needs.

3.1 Analysis

First of all, conceiving roles as first-class entities helps analysts in identifying common behaviors of the system actors, confining them into well-defined entities and associating the same roles to different actors that have some common behavior. Even if this is an advantage gained by the general role approach, in BRAIN the concept of roles as first-class entities is emphasized, thus leading analysts to reasoning consequently.

The BRAIN model helps analysts in identifying the requirements of roles. In fact, the model is based on actions and events, and analysts are required to find out which actions a (agent playing) role can perform, and which events a (agent playing) role must manage. More precisely, the proactive behavior of the agent is to be concretized into role actions, while the reactive behavior of the agent is to be translated into role events.

During the analysis, developers can exploit the XRole notation to speed up this phase. In fact, since XRole defines a structure for roles, it suggests developers what information is required to describe a role, with particular regard to actions and events. This leads developers to reason about the application not only with regard to its roles, but taking into account which actions a role can perform and what events it must handle. For example, a developer in charge of developing a bidder role for an auction, can analyze the seller and auction role (maybe provided by another developer) in order to make his role compatible and interoperable with the above two. Moreover, the availability of documents about already provided roles can help developers to analyze the existing solutions and, maybe, to reuse some of them.

3.2 Design

Depending on the set of roles and related actions/events emphasized in the previous phase, designers can, in this phase, write each role definition. The role definition implies the formalization of the set of allowed actions and events that the role can handle, providing also description about the role itself thus it can be clearly placed in the application scenario it has been designed for.

In this phase, the exploitation of the XRole notation grants several advantages. First, being based on XML, XRole provides all advantages of this language, in terms of:

- Interoperability: notations derived by XML can be used in different platforms and applications;
- Manageability: there are different tools that helps in managing XML documents;
- Automation: since XML is a structured language, its documents can be manipulated in an automatic way;
- Extensibility: XML documents can be extended depending on the needs of applications.

Then, the same XRole documents can be translated into more detailed HTML documents to suggest functionalities of the involved entities; applying different XSL sheets, the same information can be represented in different ways, adapting it to the needs of the readers. We can decide to represent many or few details of the same role, for instance because a role designer is interested in evolving roles and needs a deep knowledge of the roles themselves, while an agent designer is only interested in exploiting role features disregarding deeper details.

As sketched before, a designer could decide to extend an existing role to produce a new role more suitable for its application. Even if the XML language does not provide a simple mechanism for inheritance as object-oriented programming does, the extensibility of XML combined with appropriate tools provides several advantages, like the capability to create different views of the same information, to be extended to support future features, to be human readable, etc.[CabIS05].

3.3 Implementation

In the implementation phase, the model defined during the analysis and formalized during the design must become “real” and concretely usable. This phase is strictly tied to the interaction infrastructure the roles will be used on, since their implementation depends on the rules imposed by the bottom layer of the BRAIN framework. As already stated, there are two available interaction infrastructures for BRAIN, RoleSystem [CabLZ03b] and RoleX [CabLF03a], both written in Java, and that provides a set of classes and facilities to support the implementation of roles and their reuse across agent systems.

It must be noted that, even in this phase, XRole can support and speed up the development: the same XRole documents can be exploited to obtain the code of the roles, or at least a structure of code that can be completed by programmers. For instance, appropriate XLS sheets can translate XRole documents into Java interfaces and classes that implement the role properties. The fact that this can be performed automatically, grants two main advantages:

- Fast development: programmers save time because they can focus on the core of the code disregarding the structure in which it is inserted;

- Coherence: the structure of the code respects the design, since it is generated starting from the same information of the previous phase.

Thanks to both the facilities provided by the interaction infrastructure and the XRole notation, this phase can be successfully completed in a short time and without big efforts.

3.4 Runtime Execution

The BRAIN infrastructures provide support at runtime, enabling agents to assume, play and dismiss roles. The above-mentioned RoleSystem is an infrastructure simple to use and with good performance. RoleX, instead is more dynamic, since it is based on the manipulation of the agents to add the features of the roles; to this purpose, RoleX provides role descriptors that can be exploited by agents at runtime to decide which roles to play. Agents can also be able of understanding the XRole documents, to know the features of the described roles and to take decisions depending on them.

In general, other interaction infrastructures can be implemented inside BRAIN, respecting the model and the notation, and adding further features that can be exploited by agents during their execution in an autonomous way.

4 An Application Example

In this section we briefly report a simple example related to the game world. In a game there are well-defined interactions, which occur in a given order, and follow the game-specific laws. Here we do not focus on a specific game, but we show roles and interactions of a generic game, which can be adapted to specific games by defining the appropriate laws. Actually, several contexts in which agents interacts could be seen as instances of the generic game presented in the following.

4.1 Analysis

From the analysis emerges what follows. We identify two roles in the applications: the *player* and the game *manager*. In each game there is one manager and one or more players.

The main capability of the player is to play, but there are some collateral capabilities that are related to knowing which is the kind of the game and to leaving the game. The expected behavior concerns mainly to receive information about the game.

With regard to the manager role, the capabilities are related to notifying players about the game. On the other hand, a manager is expected to accept moves from the players, as well as reply to requests about the game situation.

4.2 Design

In the following, we report the actions and the events corresponding to the capabilities and the expected behaviors described in the previous phase. These actions and events are coded into XRole documents like the one shown in Figure 3.

4.2.1 The Player Role

Each agent that wants to participate to the game has to assume the role of *player*. The *actions* that can be performed by a player are:

- askKind. This action is used to ask the manager what kind of game it manages.
- reqPlay. The player requests the manager to play in the current game.
- move. By this action, the player tells the manager the willing of making a move. The action must contain also the description of the move.
- giveUp. It is used to notify the manager that the player leaves the current game.

The main *events* that a player must deal with are:

- notifyKind. By this event, the player is notified about the kind of the current game. This event is associated to a string that contains the actual game kind.
- accept. The player is accepted by the manager in the current game.
- requestMove. The player is requested by the manager to make a move.
- gameWon. The player is notified it is the winner of the game.
- gameOver. The player is told by the manager that the game is over.

4.2.2 The Manager Role

Each instance of the game is ruled by an agent playing the role of *manager*. The main *actions* that can be performed by a manager are:

- notifyKind. The manager notifies a player about the kind of the current game.
- accept. The manager notifies a player that it has been accepted in the current game.
- refuse. The manager notifies a player that it has been refused in the current game.
- requestMove. The manager requests a player to make a move.
- gameWon. The manager tells a player it has won the game.
- gameOver. The manager tells a player that the game is over.

The main *events* that a manager must deal with are:

- askKind. The manager is notified that a player asks for the kind of game.
- reqPlay. The manager is notified that a player has requested for playing.
- move. The manager is notified about a move of a player.
- giveUp. The manager is notified that a player wants to leave the game.

Note that the actions of the player usually correspond to the events handled by the manager, and vice versa.

4.3 Implementation

For the implementation, we have derived Java interfaces from the XRole documents defined in the previous phase. With regard to the example of the player role and its

XRole document (see Figure 3), the Java interface of its role will be like the one reported in Figure 4.

```
public interface player extends GenericRoleInterface{
   // the askKind action
   public void askKind();
   …
   // the kind of recognized events
   public static GenericEvent notifyKind    = new
                                  GenericEvent("notifyKind");
   public static GenericEvent requestMove = new
                                  GenericEvent("requestMove");
   ...
}
```

Figure 4. An example of role Java interface derived by a XRole document.

As shown in Figure 4, the interface provides the method to request the kind of the game, and a couple of events. These static events are used to make possible for the agent/role to recognize all the notified events.

4.4 Runtime Execution

To execute the application we exploit the RoleX infrastructure. This infrastructure provide different runtime services:

- Assumption of roles by agents;
- Release of roles by agents;
- List of agents playing a given role;
- Semantic description of roles and their actions/events;
- Translation from actions to events.

Incoming agents can choose the role to play, and can discover which roles other agents are playing. For instance, if another agent is playing the role of manager, no other managers are needed.

RoleX provides also a GUI that allows administrators to choose the available roles, the incompatibility among them (e.g., an administrator cannot be a player), the number of role assumption per agent (e.g., an agent cannot be a player twice), etc.

5 Related Work

Since there have been different approaches for agents based on roles, this section briefly reports those similar to the BRAIN approach; interested readers can find more details in [CabFL04h].

The Role/Interaction/Communicative Action (RICA) theory [SerOss04] aims to improve the FIPA systems [FIPA] with support for social concepts. Similarly to BRAIN, the RICA theory defines a metamodel, that can be used as a language to define communicative entities[1]; its implementation is written in Java and it is called RICA-J. RICA models the behavior of agents through roles, thus an agent is nothing more than a role player. Moreover, RICA strongly uses social concepts, thus for each standard concept, it is possible to find out the correspondent social one (e.g., roles can be specialized in social roles, actions in social actions, and so on). From this point of view, BRAIN is more general, without explicitly distinguishing between social and non social entities, since this separation strictly depends on the application context and is quite subjective.

The Aspect Oriented Programming (AOP) has been exploited to implement the concept of role by E. Kendall [Ken00]. She well describes the importance of modeling roles for agent systems. Our aim is to go beyond Kendall's considerations, and to propose roles as intermediaries for the interactions between agent applications and environments. Our concept of role is more dynamic and aims at covering different stages of the life cycle of agent-oriented applications in a more practical way, concretely supporting both the design and the implementation phases.

AALAADIN [FerG98] is a meta-model to define models of organizations. It is based on three core concepts: *agent*, *group* and *role*. Even if our approach is quite similar to the AALAADIN one, it differs for some reasons. First, we disregard the concept of *group*, while focusing on the interactions among agents and between agents and environments. Second, AALAADIN roles are tightly bound to the notion of agent, while our aim is to describe roles in a more independent way, both of applications and environments. Third, in AALAADIN, environments are mainly modeled by service agents, which is generally acceptable, but do not cover all real situations, where also agents that play roles of "pure clients" must be taken into account.

The ROPE project [Bec99] recognizes the importance of defining roles as first-class entities, which can be assumed dynamically by agents. It proposes an integrate environment, called ROPE (Role Oriented Programming Environment), which can be exploit to develop applications composed by several cooperating agents. Rather than defining an integrated but *close* environment, we aim at proposing an *open* methodology to define agent roles. We address interoperability and also the dynamic use of roles. Moreover, our definition of roles can be exploit also for interactions among agents that do not belong to the same application (i.e., are competitive); this is a relevant aspect in the design of applications for wide-open environments, such as the Internet.

The Tractable Role-based Agent prototype for concurrent Navigation Systems (TRANS) [Fou03] is a Java role system born to simulate ship navigation through an agent system. In this system, ships are represented by agents, while the kind (and the behaviour) of a ship is represented by the role the agent is playing. In TRANS, a role is defined as a normal class, thus an agent plays a role simply creating a role object and using it. An agent is free to assume the role it wants, but there are a few ties about compatibility across roles that limitate the assumption. The release of the role corresponds to the destruction of the role object itself. Being tied to a specific application scenario, TRANS results less general than BRAIN and does not provide a

[1] Comunicative entities are entities that can act in a social way, and thus can be aggregated into societies.

set of general tools and methodologies to develop role based applications, while BRAIN does.

6 Conclusions and Future Work

The BRAIN framework proposes a role-based model for agent interactions, an XML notation and interaction infrastructures. This paper has explained how the BRAIN framework, and in particular the XRole notation, can be exploited in the different phases of the development of applications composed of different agents that play roles to interact.

We can state that the main advantage of using BRAIN is that the same information is used during the whole development process, granting coherence and continuity. The XRole notation, being based on the XML language, enables to present the same information in different ways, depending on the needs of people that read such information. Moreover, XRole documents can be manipulated by automatic tools for different purpose, and even understood by agents if needed. The chance of exploiting the information about roles at runtime well suits very dynamic environments.

Currently we are investigating security models to be embedded in BRAIN and in particular in its interaction infrastructure RoleX; these models include trust and security policy ones. We are also refining the role model to make it more suitable to open and dynamic environments, supporting for example an assumption driven by prerequisites.

References

[AhuCG86] S. Ahuja, N. Carriero, and D. Gelernter, “Linda and Friends”, IEEE Computer, Vol. 19, No. 8, pp. 26-34, August 1986.

[AriLan98] Y. Aridor, D. Lange, “Agent Design Pattern: Elements of Agent Application design”, International Conference on Autonomous Agents, ACM Press, 1998.

[Bau98] J. Baumann, F. Hohl, K. Rothermel, M. Straßer, “Mole - Concepts of a Mobile Agent System”, The World Wide Web Journal, Vol. 1, No. 3, pp. 123-137, 1998.

[Bec99] M. Becht, T. Gurzki, J. Klarmann, M. Muscholl, “ROPE: Role Oriented Programming Environment for Multiagent Systems”, The Fourth IFCIS Conference on Cooperative Information Systems (CoopIS'99), Edinburgh, Scotland, September 1999.

[CabFL04a] G. Cabri, L. Ferrari, L. Leonardi, “The RoleX Environment for Multi-Agent Cooperation”, Eighth International Workshop CIA 2004 on Cooperative Information Agents, Erfurt, Germany, September 2004, Lecture Notes in Artificial Intelligence N. 3191

[CabFL04h] G. Cabri, L. Ferrari, L. Leonardi, “Agent Role-based Collaboration and Coordination: a Survey About Existing Approaches”, The 2004 IEEE Systems, Man and Cybernetics Conference, The Hague, The Netherlands, October 2004.

[CabIS05] G. Cabri, M. Iori, A. Salvarani, “Describing and Extending Classes with XMI: an Industrial Experience”, Software Evolution with UML and XML (Idea Group, Inc., Hershey-USA), Hongji Yang Editor, ISBN: 1-59140-462-2, 2005.

[CabLZ00a] G. Cabri, L. Leonardi, F. Zambonelli, “Mobile-Agent Coordination Models for Internet Applications”, IEEE Computer, Vol. 33, No. 2, pp. 82-89, February 2000.

[CabLZ00c] G. Cabri, L. Leonardi, F. Zambonelli, "MARS: a Programmable Coordination Architecture for Mobile Agents", IEEE Internet Computing, Vol. 4, N. 4, pp. 26-35, July-August 2000.

[CabLZ02a] G. Cabri, L. Leonardi, F. Zambonelli, "XRole: XML Roles for Agent Interaction", The 3rd International Symposium "From Agent Theory to Agent Implementation", at the 16th European Meeting on Cybernetics and Systems Research (EMCSR 2002), Wien, April 2002.

[CabLZ03b] G. Cabri, L. Leonardi, F. Zambonelli, "Role-based Interaction Infrastructures for Internet Agents", IEICE Transactions on Information and Systems (Oxford University Press, Oxford-UK), Vol.E86-D, No.11, November 2003.

[Cha90] R. G. Chandras, "Distributed Message Passing Operating Systems", Operating Systems Review, Vol. 24, No. 1, pp. 7-17, 1990.

[DomLD97] P. Domel, A. Lingnau, O. Drobnik, "Mobile Agent Interaction in Heterogeneous Environment", The 1st International Workshop on Mobile Agents, LNCS, Springer-Verlag (D), No. 1219, pp. 136-148, April 1997.

[FalHM99] Amal El Fallah-Seghrouchni, Serge Haddad, Hamza Mazouzi, "Protocol Engineering for Multi-agent Interaction", The 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW '99), Valencia, Spain, June 1999.

[FerG98] J. Ferber and O. Gutknecht, "AALAADIN: A meta-model for the analysis and design of organizations in multi-agent systems", The Third International Conference on Multi-Agent Systems (ICMAS), Cite des Sciences - La Villette, Paris, France, July 1998.

[FIPA] The Foundation for Intelligent Physical Agents (FIPA) web site: www.fipa.org

[Fou03] S. Fournier, D.Brocarei, T.Devogele, 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

[InvKL98] M. d'Inverno, D. Kinny, M. Luck, "Interaction Protocols in Agentis", The Third International Conference on Multi-Agent Systems (ICMAS), Cite des Sciences - La Villette, Paris, France, July 1998.

[Ken00] E. A. Kendall, "Role Modeling for Agent Systems Analysis, Design and Implementation", IEEE Concurrency, 8(2):34-41, April-June 2000.

[Lin01] Jürgen Lind, "Specifying Agent Interaction Protocols with Standard UML", The 2nd International Workshop on Agent Oriented Software Engineering (AOSE), Montreal (C), May 2001.

[OdePP01] James Odell, H. Van Dyke Parunak, Bernhard Bauer, "Representing Agent Interaction Protocols in UML", Agent Oriented Software Engineering, Paolo Ciancarini and Michael Wooldridge eds., Springer-Verlag, Berlin, pp. 121–140, 2001.

[SerOss04] Jean Manuel Serrano, Sascha 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

[Wal94] D. W. Walker, "The Design of a Standard Message Passing Interface for Distributed Memory Concurrent Computers", Parallel Computing Vol. 20, No. 4, pp. 657-673, 1994.

[Whi97] J. White, "Mobile Agents", in Software Agents, J. Bradshaw (Ed.), AAAI Press, pp. 437-472, 1997.