Agent Role-based Collaboration and Coordination: a Survey About Existing Approaches*

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 - The development of agent-based systems must take into account interactions, carefully modelling and engineering them. Roles represent a good concept that can help designers and developers dealing with interactions. So far, several role approaches for agents have been proposed. This paper presents a survey on the most important existing approaches. This survey evaluates the above approaches presenting their main characteristics and comparing them each other to find out how they provide support for analysis, design, implementation, interoperability and openness. Also the support for a formal notation is evaluated. This survey does not try to find out the best role approach, but to expose to designers and developers advantages and drawbacks of several approaches, so that they can recognize the conditions under which their use is preferable rather than others.

Keywords: Agents, Interactions, Roles.

1 Introduction

Agents are autonomous entities able to perform their task without requiring a continue user interaction [17]. Thanks to their autonomy, agents are exploited to build complex systems and applications where agents do not simply play as involved entities, but can also play on behalf of their user(s). This is true also thanks to other agent capabilities, such as mobility, which enables agents to move in a digital world made by hosts and networks, or artificial intelligence, which allows agents to learn and smartly react to the execution environment.

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, today’s applications exploit several agents at the same time, each one interacting with the others in order to achieve its goal. In such a kind of applications, often called MAS (Multi Agent Systems), coordination plays a fundamental role, since it allows agents to interact the one with the other in a productive way. Since agents are autonomous entities, coordination is not a passive task, but involves actively agents themselves, which pass from “coordinated entities” to “coordinating entities”. In other words, agents collaborate in order to successfully coordinate themselves.

Although collaboration and coordination 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 the today’s agent-based application requirements, which include the capability to operate in open and dynamic environments. To overtake the above problems, it is fundamental to use a collaboration and coordination approach able to deal with application and developer needs, and which can be applied and reused in several scenarios.

There are several approaches related to agent collaboration and coordination, including Tuple-Spaces [7], Group Computation [13], Activity Theory [18] and Roles [5]. In this paper we focus on the use of the latter, which can help designer and developers modeling the agent scenario similarly to the real life one.

Roles have been already exploited in Object Oriented approaches, where a role is defined as a set of common behaviours [12] that can be applied to an entity (often another object) in order to change its capabilities and behaviour. Other approaches promote roles as views of a particular object or entity [2], stressing the similarity between roles in computer programs and those in the real life.

Starting from previous work in Object Oriented Programming, roles have been applied to agents, which after all can be thought as autonomous and active objects, promoting the reuse of solution and making the definition of coordination scenarios easier. Roles allow not only the agent-application developers/designers to model the execution environment, but allow agents to actively “feel” the environment itself. In other words, roles allow the developer and its agents to perceive the execution environment in the same way.

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 area are security, in which we can recall the Role Based Access Control (RBAC) [19] that allows uncoupling between users and permissions, and the Computer Supported Cooperative Work (CSCW) [21], where roles grant dynamism and separation of duties. Also in the area of software development we can find approaches based on roles, especially in the object-oriented programming [8, 16], in design patterns [12], and in computer-to-human interfaces [20], which remarks the advantages of role-based approaches.

Applied to the agent scenario, roles are mainly exploited to define common interactions between agents (for instance, the interactions between the contract-net participants, or between auctioneers and bidders in an auction), and promote an organizational view of the system, which well suits agent-oriented approaches [25]. Roles embed all information and capabilities needed in a particular execution environment to communicate, coordinate and collaborate with other entities and/or agents. Thanks to this an agent (and its programmer) does not need to know details about the current execution environment but only which role to assume and use to interact with (or to exploit) the environment itself. This leads to a separation of issues related to the agent logic and its communication with other entities. The former is embedded in the agent itself, since defines the agent base behaviour, while the latter is embedded in a role and expresses an added behaviour. This separation is more emphasized at the development phase, since it is possible to develop agents and the roles they are going to use in separated time and ways, leading to a more modular development process. Another advantage of the use of roles is solution reusability. Since roles embed a behaviour applied to a specific context (for example collaboration in a MAS system), keeping it separated from agents, a set of roles can be successfully applied to other similar areas, leading not only to a faster development, but also to experience and solution reuse.

Several role approaches for agents have been proposed and implemented [5, 22, 23, 24], each one with its own characteristics and applicability. The fact that several approaches exist can disorientate the designer/developer when she has to choose a role system. In this paper we evaluate the most significant role approaches applied to agent scenarios, emphasizing their advantages and drawbacks, outlining their usability in the development of agent applications. This paper will not present a final classification of presented approaches, marking some of those as “bad” and the others as “good”. Instead, this paper proposes an analysis of different approaches to help developers understanding when and how they can be successfully used.

This paper is organized as follows: section 2 details how the role approaches have been evaluated, section 3 briefly lists evaluated approaches, section 4 reports the comparison and its results. Finally section 5 reports conclusions.

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

- Support for the application development, with particular regard to the main involved phases:
  - Analysis: this phase outlines the system requirements, disregarding implementation specific issues.
  - Design: during this phase all involved roles are defined, that means that it is decided which and how many roles will be used, which services every one will provide, how they will be assumed and released and which relationships tie them.
  - Implementation: this phase makes concrete the previous two, taking into account the exploitation of roles in the application.
- Notation: the availability of a notation to describe involved roles and their behaviour can help designer and developers to deal with roles, better understanding and applying 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.
- Run-time capabilities: today’s applications require a strong dynamism, and this is true also for agent applications, where it is required run-time adaptability to execution environment changes. Roles can grant this adaptability, providing a way to interact and collaborate with the execution environment leaving to roles themselves the task to mediate 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.
- Openness: to be widely applicable, a role approach 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.
- Interoperability: means the capability to mix the role analysis, design and implementation among different approaches, thus, for example, a designer can use the analysis phase of one approach while, during development, it can be used the development phase of another approach.

We believe the above characteristics are the main ones a role approach should focus on, and for this reason this survey will focus on them. The support to one of the above features means that the considered approach provides appropriate models or tools to help designer and developers in their work.
3 Evaluated Approaches

Due to space limitation we cannot present all approach characteristics in detail, so we simply report a brief list of evaluated approaches. We have studied all of them in order to make this survey possible. Even if these are not the all the existing role approaches, we have chosen them since we believe they are the most significant ones.

AALAADIN [10] is based on a meta-level of abstraction and defines roles as abstract representation of agent services. BRAIN [5, 6] is a multi-layer approach that defines roles by meaning of XML documents as sets of capabilities and expected behaviours. Fasli’s [9] approach defines roles as sets of obligations, rights and social commitments, focusing on the concept of sociality. GAIA [23] models a role as a set of responsibilities, permissions and activities, driving interaction by protocols. Kendall’s approach [15] defines roles as crossingcutting entities, exploiting Aspect Oriented Programming. RoleEP [22] defines roles as part of the environment where agents are executing in, and binds agents and roles each other by meaning of binding interfaces. ROPE [3, 4] defines a role as a set of permission (either required or granted) and services, modelled by service graphs. TRANS [11] defines roles as a set of services, implemented as first class entities. TRUCE [14] proposes roles as views of other agents and their services. Yu & Schmid’s [24] approach models roles as sets of rights and duties. Zhu’s [26] approach defines a role as responsibilities and capabilities, without seeing it as a first class entity but as a conjunction with an agent (role agent).

4 Comparison

This section reports the comparison among all presented approaches. Before having a look at the comparison, it can help summarizing the main characteristics of above approaches.

All approaches 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 life 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 stronger code reusability, since all approaches allow the use of the same role by different agents. A concept present in all approaches is that roles are useful for collaborative environments, where agents have a common task to achieve and collaborate to do that.

All approaches recognize that an agent can play multiple roles at the same time, but the case of a single role is treated in different ways. If in most approaches an agent is an entity able to play one role, RoleEP does not define an agent without a role. In other words, most of approaches do not explicitly limit the interaction scenario to agents that are not playing roles, recognizing still them as “agents”, while RoleEP considers agents as entities that must play at least one role. This can clash with other definitions of agents, creating mess around the concept of agent.

TRANS, Zhu’s and ROPE 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 each other.

Approaches from Yu and Shmid, Fasli and BRAIN explicitly define a role also as a set of obligations (in BRAIN they are called expected behaviour), which means that agents playing a role must behave accordingly to it. This can seem obvious, since an agent playing a specific role have to behave accordingly to it, and in fact it exploits the role services/capabilities/rights. Instead the definition of obligations is stronger, since it requires that the agent playing such role must act in a specific way depending on external events, that 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 approaches define a role as a set of rights, even if not explicitly. In fact, those approaches 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 security permissions.

Only BRAIN and AALAADIN define roles also as views of agents playing them. We believe this can be an interesting feature, since a role assumption/dismiss should change also 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 they should understand they could ask it for fuel. Here the term “understand” means that agents should be able to recognize the role played by other agents without explicitly ask 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 approaches. In fact, some approaches give a definition of the concept of role that is too generic and too far from implementation issues.

Most of the approaches define a formal notation that can help developers during the analysis phase. However, these notations are often quite complex, and can lead to a misuse (even a reject) of them. No approach seems to use the AUML [1] notation, which is explicitly conceived for agents and roles. The fact that the presented approaches do not use standard instruments (e.g., AUML), even preferring their own ones, causes a heterogeneity that cannot help developers migrating from an approach to another.

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

At the implementation phase, only BRAIN, Kendal’s and RoleEP approaches cover dynamism, that is the capability to assume/release roles at run time without static dependencies (i.e., dependencies defined in the code). This is a real important feature, since it is the only one that can make agents suitable to dynamic applications (as those for the Internet) granting a real adaptability to environment changes.

Another important thing to note, which is tied to the implementation level, is how approaches maintain roles. Since a lot of agents are mobile, and roles can help them interacting in all different scenarios where they are moving, how should roles be distributed among all platforms/sites? No approach 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 [5] helps role upgrading/refactoring in a way transparent to agents.

In our opinion it is also important to grant interoperability among different implementations of the same role system. Here “interoperability” does not strictly mean that the same implemented roles should be usable in all the implementations of the same approach, since this depends also on the implementation languages. Instead here “interoperability” is used as the capability, for agents running on a certain implementation, to “see” roles in the same way as agents running in a different implementation do. In other words, can agents share information about roles? To achieve this, a meta-level of information must be used, and only AALAADIN can provide it so far. Once again BRAIN can partially achieve it by mean of role descriptors, but they are not abstract enough.

Finally, only the TRANS approach gives the notion of priority, which defines a dependence chain among roles, and thanks to which it is possible to drive the role assumption process.

4.1 Inside the Comparison

This section summarizes the characteristics of all presented approaches. Table 1 shows support for analysis, design, implementation and other characteristics such as the capability of using roles at run-time for evaluated approaches. In this table, the symbol ● means that there is a full support, ◆ means a partial support (i.e., without roles), while ○ means that the characteristic is absent.

With regard to the support for design, analysis and implementation, Figure 1 shows a different report of the presented approaches: each approach is presented as a spot in the triangle, and its position depends on how it is implementation, design or analysis prone. Of course, approaches that support better all phases are those near the centre of the triangle, while approaches specific to only one development phase are near the corresponding corner.

As it is possible to note, only approaches that completely support the implementation phase can provide a dynamic role assumption/release. There are several approaches that do not support the implementation phase with roles, which means for example 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

<table>
<thead>
<tr>
<th>Approach</th>
<th>Support for</th>
<th>Formal Notation</th>
<th>Roles at runtime</th>
<th>Openness</th>
<th>Interoper.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>analysis</td>
<td>design</td>
<td>implementation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AALAADIN</td>
<td>●</td>
<td>●</td>
<td>◆</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>BRAIN</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>Fasli</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Gaia</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Kendall (AOP)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>RoleEP</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>ROPE</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>TRANS</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>TRUCE</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>Yu &amp; Schmid</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Haibin Zhu</td>
<td>●</td>
<td>●</td>
<td>◆</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>
produce incoherent implementation in the agent world. Furthermore, such approaches cannot apply their role model to all development phases, leading to a fragmented solution.

A formal notation, which is exploited from the most of approaches, helps in the analysis/design phase, but they should be joined with a visual notation, such as AUM. In fact, notation can be harder to read than an AUM diagram, while UML standard diagram cannot provide visual representation of the concept of role if the approach does not define it as an entity itself.

A formal notation, which is exploited from the most of approaches, helps in the analysis/design phase, but they should be joined with a visual notation, such as AUML. In fact, notation can be harder to read than an AUML diagram, while UML standard diagram cannot provide visual representation of the concept of role if the approach does not define it as an entity itself.

Figure 1. A graphical visualization of support for implementation, analysis and design of all presented approaches

It is worth noting that, as shown in Figure 1, there is no approach too much close to the design corner, which means too much design prone. In other words, approaches are generally very practical (i.e., implementation prone) or abstract (i.e., analysis prone), but never very schema-oriented (i.e., design prone).

With regard to those approaches near the centre of Figure 1, it needs to stress that they are quite recent approaches, so it is natural they are better than others, since they have been built upon other approach experiences.

4.2 Choosing the Best Approach

As written in section 1, the aim of this survey is not to find out which approach currently represents the best, but rather to help developers orienting to choose the approach that is the most suitable for their purposes. For example, BRAIN can be considered a complete approach, even if, due to the dynamism it provides, its API is more difficult to use than other approaches. Similarly, other approaches can have specific advantages and drawbacks.

If the application must be built rapidly, or it is very simple, and thus there is no need for a deep analysis, developers should choose between TRUCE and TRANS. Both these approaches grant a good separation of concerns, even if they are very practical.

In situations where the application requires a deep analysis, Fasli’s or GAIA approaches are the best ones, while Habin’s one becomes really useful if the system needs also to interact with humans.

As already written, there is no approach that particularly focuses on the design phase, and thus the choice for these situations is equally distributed among remaining approaches.

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

5 Conclusions

This paper has presented a survey of different role approaches for agents. We have chosen to present only the most representative role systems, after evaluating a lot of approaches, in order to emphasize which are the typical characteristics and behaviours.

What we found out 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 developing 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 to agent-based approaches, since they promote organizational view of the systems. So, an analysis based on roles helps in understanding the main functions of the systems, and can be easily translated in an agent-based design model.

A common limitation of the presented approaches is the lack of support for developing 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 understanding exactly which roles are required and how to implement them in a better way.

We believe that all role approaches should move in the interoperability direction, thus developers become free to exploit different approaches, each one with its own advantages and drawbacks, at different phases. This will lead to a development suitable for all agent programmers.

Acknowledgments: Work supported by the Italian MIUR and CNR within the project "IS-MANET, Infrastructures for Mobile ad-hoc Networks", and by the MIUR within the project "Trust and law in the Information Society. Fostering and protecting trust in the market, in the institutions and in the technological infrastructure".

References


