XRole: XML Roles for Agent Interaction

Giacomo Cabri, Letizia Leonardi
Dipartimento di Ingegneria dell'Informazione
Università di Modena e Reggio Emilia
Via Vignolese, 905 – 41100 Modena – ITALY
email: {giacomo.cabri, letizia.leonardi}@unimo.it

Franco Zambonelli
Dipartimento di Scienze e Metodi dell'Ingegneria
Università di Modena e Reggio Emilia
Via Allegri 13 – 42100 Reggio Emilia – ITALY
franco.zambonelli@unimo.it

Abstract

Engineering interactions is a very important issue in the design and development of Internet applications. The wideness, the openness and the uncertainty of the Internet environment call for appropriate methodologies. In this paper we propose XRole, a system that helps in dealing with such a kind of interactions in a modular and effective way. XRole is based on the definition of roles, intended as intermediaries between the application needs and the environment needs. XRole is implemented exploiting the XML language. An application example in the agent-negotiation area shows the effectiveness of the approach.

1 Introduction

The Internet world will be more and more populated by software agents, which act on behalf of users to carry out tasks. In this scenario, the interactions among agents are becoming an important issue that must be taken into consideration at the different phases of the application development [Cabri, 2001]. On the one hand, complex goals are faced by multi-agent systems by dividing the main goal in several simpler goals, each one assigned to one agent; this lets emerge the need for making agent of the same application interact for cooperating to carry out the global task. On the other hand, the Internet is becoming a place where resources and services for agents are available, but often agents have to compete to achieve them; this points out that interactions between agents of different applications are to be dealt with.

We propose to deal with agent interactions by a three-level model (see Figure 1), thought to manage both interactions among agents of the same application and interaction between agents of different applications. In this model, the application level is represented by the agents; the lowest level concerns the environment, which defines its own policies to rule agent-to-agent and agent-to-resources interactions. The middle level is the one focused by this paper, and concretely enables the interactions. To achieve flexibility and dynamism, we propose to center it on the concept of *role*.

In the agent scenario, a *role* is defined as the *behavior* and the *set of the capabilities* expected for the agent that plays such role [Kendall, 2000]. This leads to a twofold viewpoint of the role: from the environment point of view, the role imposes a defined behavior to the entities that assumes it; from the application point of view, the role allows a set of capabilities, which can be exploited by agents to carry out their tasks. There are some characteristics of roles that lead to deal with them separately 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 and then for different agents. Finally, roles are *related to contexts*, which means that each environment can impose its own rules and can grant some local capabilities, forcing agents to assume specific roles. As mentioned before, roles represent behaviors that agents are expected to show; who expects such behavior are entities external to agents themselves, mainly organizations [Zambonelli et al., 2001] and environments.

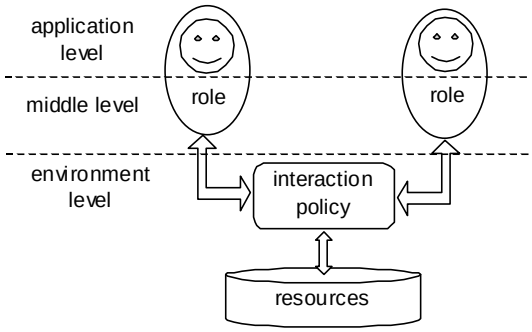

Figure 1. Role-based interaction model

The contribution of this paper is to propose a system, called XRole, which enables the definition of roles for agents, in a way that is suitable to be exploited at different phases of the life cycle of an agent-based application. Such system is based on XML to exploit all advantages this language provides. The aim of this system is helping the definition of the interactions among agents and supporting their implementation.

The paper is organized as follows. Section 2 presents the XRole system. Section 3 shows an application exam-

ple, which exploits XRole to define and use roles. Section 4 presents the related work. Section 5 concludes the paper and sketches some future work.

2 XRole

In the context of the interaction among agents, we propose XRole, a system for the definition of roles for agents.

2.1 Why XML

The incredible success of HTML has led the WWW consortium to the development of XML [W3C, 1998], a language for data representation that is likely to become a standard for data interoperability in the Internet, due to the advantages it can provide in this context. In fact, XML represents data in a familiar (HTML-like) tagged form, and explicitly separates the treatment of data from its representation. This achieves both the well-appreciated feature of human-readability and the platform-independence required for the Internet. In addition, XML can be made capable of representing whatever kind of data and entity one is likely to find in the Internet: documents, services and objects, as well as agents. These characteristics let us think that interoperability in the Internet will be information-oriented and based on XML, rather than service-oriented as may happens in architectures based, for instance, on CORBA [OMG, 1997].

By using XML, the description of each role can be also presented to human people via an appropriate XSL sheet that transforms the information in a human-understandable document, such as a HTML page. This lets programmers develop their agents knowing which the available roles are, for example by searching for appropriate roles in a repository.

Moreover, if the XML documents follow appropriate rules, they can be managed also by automated tools, and by the agents themselves, which can be enabled to understand the content of a XML documents and exploit or manipulate it without the need of the intervention of human people.

With regard to role catalogues, we point out that XML carries great advantages in this context, since the same XML document can be exploited for both the description of a role and the operative use of it.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT role (name, description?, keyword*,
action*)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT description (#PCDATA)>
  <!ELEMENT keyword (#PCDATA)>
  <!ELEMENT action (description, ret_value?,
act_name, parameter*)>
       <!ELEMENT ret_value (#PCDATA)>
       <!ELEMENT act_name (#PCDATA)>
       <!ELEMENT parameter (par_name, type)>
       <!ELEMENT par_name (#PCDATA)>
       <!ELEMENT type (#PCDATA)>
```

Figure 2. The DTD followed by XML roles defined in XRole

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
elementFormDefault="qualified">
 <xsd:element name="act_name"
type="xsd:string"/>
 <xsd:element name="action">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element ref="description"/>
    <xsd:element ref="ret_value" minOccurs="0"/>
    <xsd:element ref="act_name"/>
    <xsd:element ref="parameter" minOccurs="0"
maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="description"
type="xsd:string"/>
 <xsd:element name="keyword" type="xsd:string"/>
 <xsd:element name="name" type="xsd:string"/>
 <xsd:element name="par_name"
type="xsd:string"/>
 <xsd:element name="parameter">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element ref="par_name"/>
    <xsd:element ref="type"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="ret_value"
type="xsd:string"/>
 <xsd:element name="role">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element ref="name"/>
    <xsd:element ref="description"
minOccurs="0"/>
    <xsd:element ref="keyword" minOccurs="0"
maxOccurs="unbounded"/>
    <xsd:element ref="action" minOccurs="0"
maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="type" type="xsd:string"/>
</xsd:schema>
```

Figure 3. The XML Schema followed by XRole roles

2.2 Definition of Roles

In XRole, roles are defined by XML documents that respect a well-defined DTD (see Figure 2). Alternatively, a XML Schema can be used, such as the one reported in Figure 3. By analyzing both the DTD and the XML Schema reported, one finds the main elements we have identified to describe a role. They are:

- name. The name given to the role. It would be useful that the name was unique, to identify a precise role all over the world. But we agree that it is a too strong requirement, so we envision that a name is valid in a given context (i.e., an application area, an Internet domain, and so on).
- description. A high-level description of the role. Such description is useful for designers to understand which the aim of the roles is, in human-readable sentences.

- keyword. Zero or more keywords can be used to identify the role. They are useful for human people, automated tools and also agents in the search of roles matching given criteria.
- action. It describes actions available to the agent if it assumes this role, in order to interact with other entities, such as other agents or execution environments. This description is quite general, specifying a *name*, a *description*, a *return value*, and a *list of parameters*.

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.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl"
href="role2html.xsl"?>
<role
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="roleSchema.xsd">
 <name>MyRole</name>
 <description>example of role</description>
 <keyword>example</keyword>
 <keyword>XRole</keyword>
 <action>
  <description>interact with
someone</description>
  <ret_value>void</ret_value>
  <act_name>talk_with</act_name>
  <parameter>
   <par_name>receiver</par_name>
   <type>AgentID</type>
  </parameter>
  <parameter>
   <par_name>message</par_name>
   <type>string</type>
  </parameter>
 </action>
</role>
```

Figure 4. An example of role defined in XRole

2.3 Using Roles

We emphasize that XRole is not bound to a given agent system or to a given interaction infrastructure. Instead, thanks to the high degree of interoperability provided by XML, it can be exploited at different phases and by different systems. In this section we show how XRole can be exploited at the different phases of the software production.

Design phase

The description of roles can be useful at the design phase, because gives the designers useful information about the agents that may compose the application under development. Like design patterns [Aridor and Lange, 1998], the description does not provide code or libraries, but high-level suggestions that can be exploited at this stage of the software life cycle. In addition, sets of related roles can be defined, in order to provide a group of interacting roles to easily build up an application; in fact, each set can define not only the roles belonging to it, but also the relationships among the entities playing such roles. More than bare patterns, XRole descriptions can be automatically translated into actual code, as shown in the next subsection.

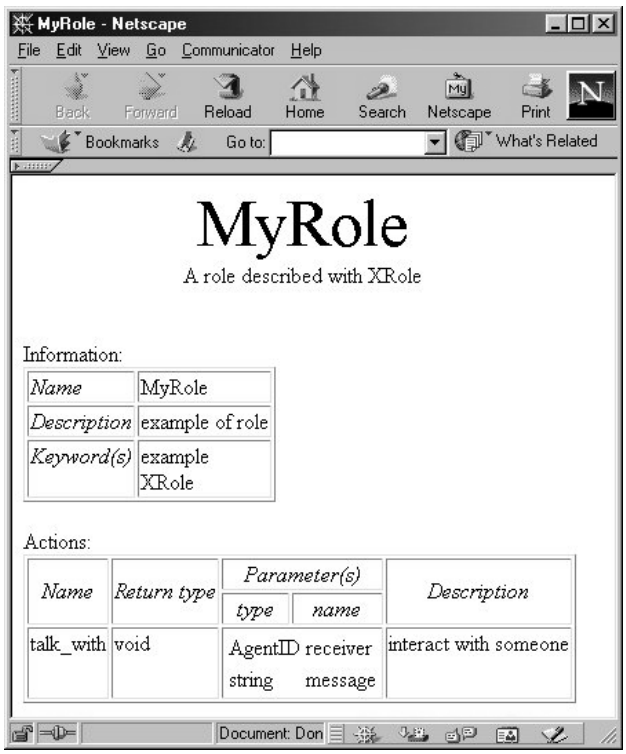

Figure 5. A human-readable description of an XRole role

Thanks to the adaptability of XML, a role defined in XRole can be presented to the designers in different ways, to meet different requirements. The simplest way is to let an appropriate XSL sheet produce a HTML document that presents the information about the role. Figure 4 reports a simple role defined in XRole that will be assumed as an example in the following of the section.

Starting from such role, the HTML document reported in Figure 5 can be created and published. More sophisticated views can be generated by appropriate XSL sheets, which tell the designers what they need. For example, a HTML documents can be created, which lists all roles that refer to the same keyword, or implement a given action.

Implementation phase

Using languages that support the notion of interface, such as Java, it can be very simple to derive an interface from a given role defined in XRole. For example, by using the XSL document reported in Figure 6, the role defined in Figure 4 can be translated into the Java interface shown in Figure 7, where the name element of the role defined in XRole is used as the name of the Java interface.

Moreover, situations where one agent plays different roles are usual. Also in this case, defining the roles in XML is very useful, because they can be combined by an appropriate XSL document.

Runtime

From the hosts' point of view, a hosting server that accepts agents using a given role must provide for the ac-

tual implementation of the actions defined in such role. For instance, referring to the role depicted in Figure 4, the server must somehow implement the action talk_with, that is, the talk_with method of the interface MyRole of Figure 7.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--XRole 0.1-->
<!--Transform a role into a Java interface-->
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format"
exclude-result-prefixes="xml">
 <xsl:output omit-xml-declaration="yes"
method="text" indent="yes"/>

 <xsl:template match="role/keyword">
 <xsl:text>&amp;quot;</xsl:text>
 <xsl:apply-templates/>
 <xsl:text>&amp;quot;</xsl:text>
 <xsl:if test=". != /role/keyword[position() =
last()]">
   <xsl:text>, </xsl:text>
  </xsl:if>
 </xsl:template>

 <xsl:template match="role/action/parameter">
  <xsl:value-of select="./type"/>
  <xsl:text> </xsl:text>
  <xsl:value-of select="./par_name"/>
  <xsl:if test=". != ../parameter[position() =
last()]">
   <xsl:text>, </xsl:text>
  </xsl:if>
 </xsl:template>

 <xsl:template match="role/action">
  <xsl:text>
 /* </xsl:text>
  <xsl:value-of select="description"/>
  <xsl:text> */
 </xsl:text>
  <xsl:value-of select="ret_value"/>
  <xsl:text> </xsl:text>
  <xsl:value-of select="act_name"/>
  <xsl:text>(</xsl:text>
  <xsl:for-each select="parameter">
   <xsl:apply-templates select="."/>
  </xsl:for-each>
  <xsl:text>);</xsl:text>
 </xsl:template>

 <xsl:template match="role">
  <xsl:text>public interface </xsl:text>
  <xsl:value-of select="name"/>
  <xsl:text>
{
 String description = &amp;quot;</xsl:text>
  <xsl:value-of select="description"/>
  <xsl:text>&amp;quot;;
 String keyword[] = {</xsl:text>
  <xsl:for-each select="keyword">
   <xsl:apply-templates select="."/>
  </xsl:for-each>
 <xsl:text>};
 </xsl:text>
  <xsl:for-each select="action">
   <xsl:apply-templates select="."/>
  </xsl:for-each>
  <xsl:text>
}</xsl:text>
 </xsl:template>
</xsl:stylesheet>
```

Figure 6. The XSL document from XML to Java interfaces

However, XRole does not define the ways methods are implemented, and each architecture can choose the most appropriate one.

In a dynamic environment agents can search for an appropriate role by using a XML query language [Deutsch et al., 1999]. In addition, agents can dynamically search for appropriate roles that are needed, and assume the found roles at runtime. This is permitted by the fact that valid XML documents can be managed in an automated way, in this case by agents themselves, which can understand the description of a role decide whether assume or not such role, on the basis of different criteria.

```
public interface MyRole
{
 String description = "example of role";
 String keyword[] = {"example", "XRole"};

 /* interact with someone */
 void talk_with(AgentID receiver, string
message);
}
```

Figure 7. The XRole example translated into a Java interface

3 Application Example

In this section we present an example of application where interactions are designed following the three levels of the model introduced in the previous section.

An interesting negotiation means in the Internet context is the *auction*. In an auction there are entities that make resources available and entities that are interested in using/acquiring such resources. The former ones are usually called *sellers*, while the latter ones are called *bidders*. Usually, there is an intermediate entity, called *auctioneer*, which actually performs 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. The seller can set a *reserve price*, i.e., a price under which it does not want to sell the resource.

There are several different forms of auction, depending on the number of participants, on the criteria with which the resources are assigned, and so on. We focus on the auctions with one seller and multiple bidders at a time, ruled by several mechanisms: for example, English, Dutch, first-price and Vickery [Agorics, 1996].

In the following we show how XRole can be exploited to describe roles that can be assumed by agents of auction applications. The reported XML documents are quite simple (due to the paper length limitation), but they aim at giving an idea of the features of XRole, and can be extended to be used in a real application.

The following role can be defined as “standard” by a set of auction sites, which either make the corresponding XML documents available to designers/developers or point to a “repository” from which roles can be retrieved. As previously stated, these roles can be part of a set of roles related to auction applications, which defines also the relationships among roles. The implementations of

the actions are then delegated to each site accepting agents that play these roles.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl"
href="role.xsl"?>
<role
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="roleSchema.xsd">
 <name>bidder</name>
 <description>an agent attending an auction,
which can bid to achieve a good on
sale</description>
 <keyword>auction</keyword>
 <keyword>bidder</keyword>
 <action>
  <description>make a bid</description>
  <ret_value>void</ret_value>
  <act_name>bid</act_name>
  <parameter>
   <par_name>good_id</par_name>
   <type>string</type>
  </parameter>
  <parameter>
   <par_name>bid</par_name>
   <type>int</type>
  </parameter>
 </action>
 <action>
  <description>ask for the status of an
auction</description>
  <ret_value>int</ret_value>
  <act_name>ask_status</act_name>
  <parameter>
   <par_name>good_id</par_name>
   <type>string</type>
  </parameter>
 </action>
</role>
```

Figure 8. A *bidder* role defined in XRole

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE role SYSTEM "roleDTD.dtd">
<?xml-stylesheet type="text/xsl"
href="role.xsl"?>
<role>
 <name>seller</name>
 <description>an agent attending an auction,
puts a good on sale</description>
 <keyword>auction</keyword>
 <keyword>seller</keyword>
 <action>
  <description>put a good on sale</description>
  <ret_value>void</ret_value>
  <act_name>put_on_sale</act_name>
  <parameter>
   <par_name>good_id</par_name>
   <type>string</type>
  </parameter>
  <parameter>
   <par_name>description</par_name>
   <type>string</type>
  </parameter>
  <parameter>
   <par_name>reserve_price</par_name>
   <type>int</type>
  </parameter>
 </action>
</role>
```

Figure 9. A *seller* role defined in XRole

The first important role in an auction is the *bidder*. This role is assumed by an agent whose willing is to buy a resource on sale. In Figure 8 is reported the XML document conforming to XRole, which describes the role of the bidder.

In this example, two main actions are allowed for the agent that assumes such role: the bid, which is used to make a bid and the ask_status, which asks for the status of an auction.

The second significant role in the auction context is the *seller*. Figure 9 shown how a seller role can be described by XRole.

In this example, an agent that assumes the role of seller is allowed (and also is expected) to put a good on sale by means of the put_on_sale action, specifying information about the good and the minimum price it wants to earn.

4 Related Work

Even if it has not been designed in connection with roles, Aspect Oriented Programming (AOP) seems to provide interesting mechanisms to support the management of roles for agents [CACM, 2001]. AOP starts from the consideration that there are behaviors and functionalities that are orthogonal to the algorithmic parts of the objects [Kiczales et al., 1997]. So, it proposes the separate definition of components and *aspects*, to be joined together by an appropriate compiler (the *Aspect Waver*), which produces the final program. The separation of concerns introduced by AOP permits to distinguish the algorithmic issues from the behavioral issues. Since an aspect is "*a property that affect the performance or semantics of the components in systemic way*", it is evident the similarity with a role. Differently from XRole, AOP is thought to achieve performance and maintainability in software development. In our opinion, even if the AOP is very similar to XRole, one possible limitation is its lack of flexibility in the definition and usage of aspects/roles. This is due to the fact that AOP focuses on software development rather than addressing the uncertainty and complexity issues of wide-open environments, such as the Internet. From this point of view XRole differs from AOP because it exploits the high degree of interoperability given by XML.

The AOP has been exploited to implement the concept of role by E. Kendall [Kendall, 2000]. 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 [Ferber and Gutknecht, 1998] is a metamodel 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 [Becht et al. 1999] 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, XRole aims at proposing an *open* methodology to define agent roles. It addresses interoperability and also the dynamic use of roles. Moreover, XRole can be used to define roles 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.

5 Conclusions and Future Work

This paper has presented XRole, a XML-based system designed to define roles for agent applications. The exploitation of the XML language gives flexibility and interoperability to systems, and allows exploiting the definition of roles at different phases of the life cycle of applications.

Some research directions for future work may be the following.

First, the implementation of XRole is in progress, and some issues related to the actual implementation of roles are to be faced. The fact that an agent can assume a role at runtime must be carefully evaluated, and the related implementation may require appropriate mechanisms and constructs, possibly provided by the implementation language.

Second, it could be interesting the definition of "repositories" of roles, from which agents can chose the more appropriate for their tasks. Which could be the most appropriate technology to create such repositories? And which access policies must be defined? If each repository is seen as a resource, meta-roles could be defined to rule the access to them. Moreover, the fact that agents can assume roles dynamically at runtime, imposes to resolve the issues sketched in the previous paragraph, to make effective an approach based on role repositories.

Finally, we are going to test XRole in applications different from the auction-related area, to verify its applicability in a wide range of application areas.

Acknowledgements

Work supported by the NOKIA Research Center Boston, by the Italian MURST within the project "MUSIQUE" and the Italian Research Council (CNR).

References

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

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

[Becht et al. 1999] 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, Sept. 1999.

[Cabri, 2001] G. Cabri, "Role-based Infrastructures for Agents", The 8th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS 2001), Bologna (I), October 2001.

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

[Deutsch et al., 1999] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Maier, and D. Suciu, "Querying XML data", Bulletin of the Technical Committee on Data Engineering, Vol. 22, No. 3, pp. 27-34, 1999.

[Ferber and Gutknecht, 1998] 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'98), 1998.

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

[Kiczales et al., 1997]G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. M. Loingtier, J. Irwin, "Aspect-Oriented Programming," Xerox Corporation, 1997. http://www.parc.xerox.com/csl/projects/aop/

[OMG, 1997] OMG, "CORBA 2.1 specifications", 1997, http://www.omg.org.

[W3C, 1998]WWW Consortium, eXtensible Markup Language (XML) 1.0, W3C Recommendation, http://www.w3.org/TR/REC-xml, 1998.

[Zambonelli et al., 2001] F. Zambonelli, N. R. Jennings, M. Wooldridge, "Organizational Rules as an Abstraction for the Analysis and Design of Multi-agent Systems", International Journal of Software Engineering and Knowledge Engineering, 11(3):303-328, 2001.