

# Binding Agent Roles to Environments: the R4R approach

Luca Ferrari

*Dipartimento di Ingegneria dell'Informazione  
Università degli Studi di Modena e Reggio Emilia  
Modena, Italy, 41100  
ferrari.luca@unimore.it*

## ABSTRACT

*Agents represent a great technology that allows developers to build complex applications. Agents' capabilities increase as much as they can cooperate and coordinate together in Multi Agent System. Developers can exploit role-based approaches to deal with coordination and cooperation issues. In this paper a new approach to the role development and deployment is presented: thanks to adoption of a clear model and of Aspect Oriented Programming, the role development and deployment is simplified guaranteeing the respect of any environment specific rule.*

**KEYWORDS:** Mobile Agents, Roles, Aspect Oriented Programming, Agent Coordination, Resources

## 1. INTRODUCTION

Agents are autonomous computational entities with problem-solving capabilities, that can perform their task(s) in open and dynamic environments [17]. Thanks to their main properties (autonomy, reactivity, sociality and mobility) agents can be successfully exploited in today's complex systems, even playing as digital human counterparts. Today's agent-based systems often involve multiple agents in order to divide complex tasks into simpler ones; such kind of applications are called MAS (Multi Agent System). In MAS systems coordination among agents is fundamental and several approaches exist to manage it, including Tuple-Spaces [10], Group Computation [16], Activity Theory [19] and Roles ([11], [12], [21]). Of the previous approaches, those that seem the most appropriate to face the dynamism required by MAS systems are those based on *roles*.

Roles represent stereotypes of behavior that agents can use in order to achieve their aims [3]. The idea is to abstract some behaviors common to different agents, embedding them into one or more roles, that agents can assume and use on demand during their life in order to express the above be-

havior. This leads to a clear separation of concerns during the agent development: roles and agents can be developed in separated and different ways, and even at different times, promoting solutions and knowledge reuse [3].

Figure 1 shows an overview of what happens on an environment that hosts agents. The host will run an *Agent Platform* (AP), that is a container where agents live and that provides access to the surrounding environment. If the AP supports also agent mobility, it is called *Mobile Agent Platform* (MAP). Agents living in the platform can assume available roles (represented as dresses in Figure 1), thus changing their behavior and appearance to other agents. Roles can be used to enhance intra-agent communications, as well as agent-platform communication, or to simply improve and enrich the agents capabilities. In other words, agents playing a role (i.e., *dressed* in Figure 1) can better interact within the surrounding environment than plain (i.e., *naked*) agents.

Roles must face portability issues: they should provide the same facilities on each AP they are installed into, especially in the case of MAP, where agents can migrate among different hosts.

This paper shows the Resources For Roles (R4R), a system that promotes the use of *resource*s as integration points between a role and an environment. Such points are used to apply *environmental rules* defined at the time of the *role deployment*.

In the rest of the paper, the term *environment* will be used to represent either the surrounding environment of the AP (the host), as well as the application context of a MAS application.

The paper is structured as follows: section 2 introduces more role related concepts and motivates the adoption of resources and rules; section 3 shows details of the currently available R4R implementation and presents a concrete application example; section 4 compares our approach to other similar approaches; finally section 5 concludes this paper.

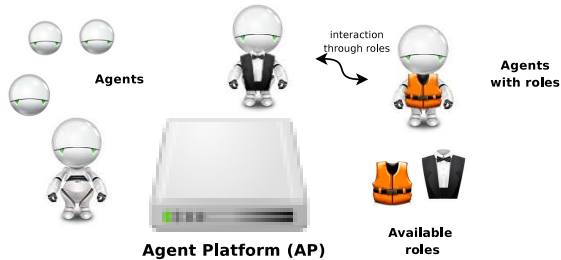


Figure 1: An overview of agents running on a specific host an playing roles.

## 2. CONCEPTS AND DEFINITIONS

This section introduces the main role concepts, motivating the adoption of resources and environmental rules.

### 2.1. Roles and their application

The importance of the adoption of roles in multi-agents and mobile-agents applications have been recognized and emphasized by several research activities, leading to different role proposals and system implementations ([4], [12]). All such proposals and approaches mainly differs in the way a role is defined and, most notably, on how a role can be used by agents. In particular, the degree of dynamism and coupling between agents and roles, either during the application life-cycle or its development, allows a classification of each role approach [4]. One concept all role approaches converge to is that a role is a *capability enabler* for the agent that is playing (i.e., has assumed) it.

BRAIN (Behavioural Role Agent Interactions) is a very flexible role approach that allows agents to dynamically exploit roles, letting the latter to be developed in several ways [7]. BRAIN defines a role through a set of capabilities and an expected behavior. The former is implemented through a set of role actions, while the latter through a set of events, that the role is supposed to manage in order to behave in a specific way (see definition 1). Managing events through the assumed role, makes the agent behaving accordingly to the role specification [7].

$$role = (actions, events) \quad (1)$$

It is important to note that, disregarding the role approach, a role must be developed accordingly to a specific environment [3]. For example, roles can be developed to embed a part of a communication protocol for a specific context, as for agents participating in an e-democracy scenario [6] or in an Ambient Intelligence scenario [5]. It is clear that, being roles tied together and to the context, they must be developed with regard each other and the application environment. Some of the role actions will be environment-

independent, being tied only to the application context, while others will be environment-dependent, tying thus the role implementation to the environment where it will be adopted. The definition 1 can be therefore specialized in the 2.

$$role = \begin{pmatrix} actions_{environment-independent}, \\ actions_{environment-independent}, \\ events \end{pmatrix}$$

$$action_{environment-dependent} = \begin{pmatrix} action, \\ environment \end{pmatrix} \quad (2)$$

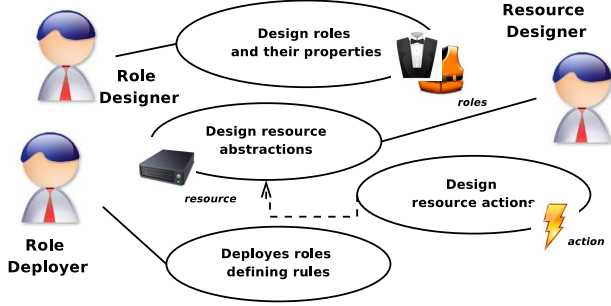
### 2.2. From plain roles to resource based roles

In order to limit the impact of the environment-dependent actions, that undermine the role portability, the R4R approach exploit the concept of *resources*, abstractions over environment entities. A resource can be a physical device (e.g., a printer), a service (e.g., a database cluster), another couple agent-role (e.g., an agent providing a yellow-pages service) or a mix of them. Typically resources are used to embed an environmental-dependent entities, and a set of actions that can be performed through them. The role definition is therefore revisited, to include also a set of resources (see definition 3). The same resource can be exploited by different roles, exposing different actions to different roles. For example, a shared `database_resource` can expose only a `query` action to a `database_user` role, while can provide actions like `create_db`, `drop_db`, `manage_usr` to a `database_administrator` role.

$$role = \begin{pmatrix} actions_{environment-independent}, \\ events, \\ resources \end{pmatrix}$$

$$resource = \begin{pmatrix} actions_{environment-dependent}, \\ policies \end{pmatrix} \quad (3)$$

Since resources are tied to a specific environment and a context, they cannot be accessed in an not discriminated way, but their use must happen through a set of *resource policies*. These policies define how the resource can be used by roles (and therefore by agents), implementing a set of rules specific for each application context. For instance, in the above database the `database_user` role can exploit the `query` operation only after having obtained access to a database connection. The database connection can be obtained only following the rules of the application environment, that can impose some pooling techniques, rather than others. To achieve this, there must be a `database` resource that is exploited by the `database_user` role and is accessed through *resource policies*. From this simple example, it should be clear that the roles actions should exploit



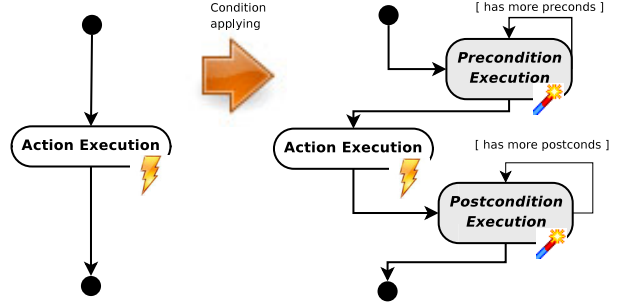
**Figure 2: Different actors involved in the role development and deployment.**

resources and that should be *decorated* with a set of other actions (resource policies) to improve the role integration within the environment it is being played. Manually decorating every single role with environment rules will lead to a static dependency between the role and its environments, that is not allowed in dynamic scenarios. There is the need of an automatic way to integrate roles into environments, applying policy rules to all resources a role exploits. If roles are designed exploiting resources, the application of policy rules can be performed at the *role deployment* time. Please note that the actors involved in the design/development phases (see Figure 2) are the followings:

- *resource designer*: designs a set of resources that abstract some environment dependent entities, and that can be used in one or more roles;
- *role designer*: is in charge of designing a role pattern that express the stereotype of a behavior, using resources when needed;
- *role deployer*: defines the rules that a specific role action must follow when accessing one or more resources.

### 2.3. Definition of policies: pre and post conditions

Environmental policies can be expressed as a set of *preconditions* and *postconditions*, respectively a set of conditions that must be validated before and after the execution of the role action. The conditions could be bound to every action available through a resource (see definition 4). Despite their name, pre and post conditions could be not only simple tests, but also concrete actions that must be (successfully) executed before and after a role action. This means that the execution of a role action has a modified control flow (see Figure 3) that force the action to be executed only after all the preconditions and before all the postconditions.



**Figure 3: A role action control flow is modified through pre and post conditions.**

In the database example of the previous section a precondition could check the database connection against a maximum allowance, as well as a postcondition could force the immediate database disconnection in order to keep the service availability for other incoming agents. Another example could be an intra-agent communication, that could force a postcondition of logging, in order to realize an overheard service for automatic role suggestion [8].

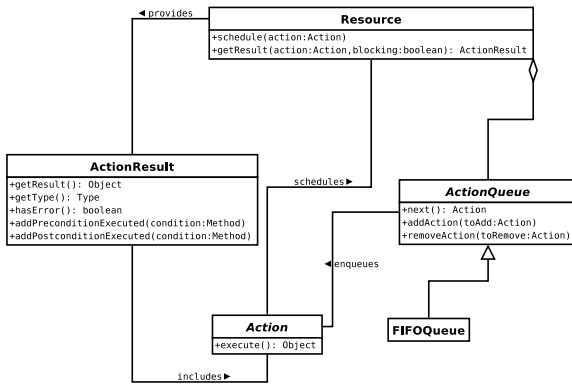
$$resource = \left( \begin{array}{c} actions_{environment-dependent}, \\ preconditions, \\ postconditions \end{array} \right) \quad (4)$$

The conditions are defined by the role deployer actor, that knows how to effectively bind a role to the environment she administers. A condition can execute any piece of code, even if not belonging to any role or resource. This is very important to implement environment policies, since the role deployer can thus bind a role action to a library that only that specific environment is using (e.g., a third party library). This also promotes the code and solution re-usability, since each environment is free to implement and adopt their own libraries.

Each condition must include the information about the resource action it refers to, the context (pre or post) and, of course, the condition to be checked or the action to be executed (see definition 5). Additionally, preconditions specify if the execution must continue in case of failure.

$$condition = \left( \begin{array}{c} role\_action, \\ \{pre|post\}, \{check|action\}, \\ [continue\_if\_fails] \end{array} \right) \quad (5)$$

The adoption of pre and post conditions enhances the expressiveness of the environment policies. In fact, thanks to preconditions it is possible to even inhibit the role action execution (e.g., the resource is, at the moment, under maintenance), while thanks to postconditions it is possible to alter the result of a role action in the surrounding environment (e.g., changes to a just inserted record in a database). Moreover, postconditions could be used to implement a *re-*



**Figure 4: Main classes of the resource/action abstraction.**

*action* to a role action, changing the action impact on the surrounding environment, and make the latter reacting to a specific role action. For instance, it is possible to react to an *insert* role action against a database performing an automatic database replication to another environment, or notifying other agents of the availability of a new record in a way very similar to that of a reactive tuple space [9]. Conditions can be used to perform the role deployment and integration with the environment.

### 3. IMPLEMENTATION DETAILS

This section shows the R4R implementation, that is available under the GNU GPLv3 licence and has been implemented in Java as a module for the Aglets Mobile Agent Platform [1], even if it does not depend in any way from such platform.

Even if R4R does not depend from any particular role approach, it has been implemented to be compatible with the RoleX role system [7], an interaction infrastructure of BRAIN.

R4R exploits the Aspect Oriented Programming (AOP) [15], with particular regard to AspectJ [2], and the Java Annotations to implement the conditions for role actions.

#### 3.1 Resources

From the design point of view a resource is quite simple: it must allow one or more actions, that must be scheduled and executed depending on the resource implementation details. In order to be as much portable as possible, a resource includes an *action queue*, that stores incoming actions and sorts them depending on different strategies (e.g., priority, FIFO, LIFO, etc.). There is no assumption on the threading system, letting to the resource designer the task of defining how to manage requests.

```

public interface Action {
    public Object execute()
        throws Exception;
}

public class ActionResult {
    public Object getResult();
    public Type getType();
    public boolean hasError();
    public void addPreConditionExecuted(
        Method method);
    public void addPostConditionExecuted(
        Method method);
}

```

**Figure 5: The Action and ActionResult declaration.**

Figure 4 shows the main classes used to implement the resource structure. The main class is Resource, that is composed by an ActionQueue, that stores Actions to be executed. The Action interface is quite simple (see Figure 5): it only has the `execute()` method that can be used to wrap what the action must do. The action return value is embedded into an ActionResult object (see Figure 5), a wrapper that provides additional information on the action execution, for instance occurred error(s).

At glance Resources accept Actions to be scheduled and executed, and return ActionResult accordingly. The main method of Resource are (see Figure 6):

- `scheduleAction(..)` accepts an action, checks if the action is valid for the resource and enqueues the action for processing it;
- `getResult(..)` provides the result of a previously scheduled action, either in blocking or non-blocking way.
- `getAllowedActions()` provides a set of actions that the resource can accept and schedule.

Please note that in this phase, no pre or post conditions definition are required or involved.

#### 3.2 Roles and Actions

The R4R approach defines that resources must be accessed through roles. For this reason, the RoleX tagging interface GenericRole [7] is implemented by a concrete class Role that provides support for resources (see Figure 7).

It is important to note that resource actions should not be executed directly, but through the `execute(..)` method of the Role class. This is due to allow the role to enforce

```

public class Resource<A extends Action>
implements Runnable{
    ActionQueue<A> scheduledActions = null;
    public boolean scheduleAction(A action);
    public ActionResult getResult(A action,
        boolean blocking);
    public Set<A> getAllowedAction();
    ...
}

```

**Figure 6: The Resource class.**

```

public class Role
implements GenericRole{
    public Set<Resource<TaggedAction>>
        getResources();
    public Set<TaggedAction>
        getResourceActions();
    public ActionResult
        execute(TaggedAction action)
        throws Exception;
}

```

**Figure 7: The Role class.**

the action control flow, for instance checking required privileges, etc.

Please note that the role of Figure 7 does not work directly with Actions, but with their specialization TaggedActions, that are actions with an OperationDescriptor associated to it. The latter is a semantic descriptor of an action that can be used to understand the meaning of an action execution (for further details see [7] and [3]).

The adoption of OperationDescriptors allows an high degree of dynamism: agents can search for a specific action only with semantic information, disregarding implementation details (like knowing which resource an action must be scheduled to).

### 3.3 Implementing conditions: the Resource Scheduling

Having designed and implemented the resources and the roles required, it is the moment to deploy them to a particular environment, and here is where the pre and post conditions come. Pre and post conditions are applied modifying the resources' action scheduling: all preconditions are executed before the resource action, while all the postconditions are executed after it. This creates a set of *scheduling rules* specific for each resource action.

```

public @interface Precondition {
    String className() default "this";
    String methodName() default "";
    boolean continueIfFails() default false;
    String actionClassName() default "";
}

```

**Figure 8: The @Precondition annotation.**

As stated before in this section, the above scheduling rules are implemented using Aspect Oriented Programming, and in particular the AspectJ [2] language extension. Nevertheless, deployers are not required to write their own scheduling changes programmatically, rather they can define pre and post conditions in a declarative way. To achieve this, appropriate Java annotations have been defined.

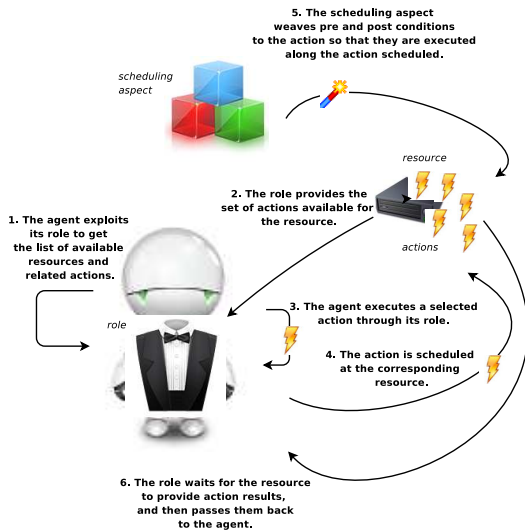
Preconditions are defined through the @Precondition annotation, that is reported in Figure 8. As readers can see, the annotation includes information about the class and method to invoke as precondition (respectively `className()` and `methodName()`), information about aborting due to errors (`continueIfFails()`) and the action to which this precondition refers to (`actionClassName()`). The latter is used when the precondition scope must be limited to a specific action (and thus to one or more roles). Postconditions have a similar annotation, @Postcondition, that is not shown for brevity.

The condition processing is performed through the Scheduling aspect. This aspect captures a specific set of join-points [2] and implements *before* and *after* advices to execute each declared pre and post condition. The execution of the action is therefore surrounded by the above advices, that change the control flow as reported in Figure 3.

R4R allows two main ways of declaring pre and post conditions through annotations:

- *declaring them within the action code*: is the less portable method, since the action code must be manually tagged with pre and post conditions. Nevertheless, this is the most customizable approach: each action can specify its own rules;
- *declaring them where the condition code is required*: is the most practical way and also the most portable. Nevertheless, in this way it is not possible to define per-action different rules.

Please note that, in the case of multiple pre or post conditions, a single ActionResult is returned; such result is shared among conditions and thus represents a global result over the execution of all conditions.



**Figure 9: A graphical overview of the whole mechanism.**

Figure 9 shows the complete overview of the process of executing environment specific actions through resources in the R4R approach:

1. the agent obtains, through one of its roles, a list of actions available on the resources owned by such role itself;
2. the role queries each resource and returns a list of available actions;
3. the agent chooses an action (depending on its semantic description) and requires to the appropriate role to execute it;
4. the role schedules the action to the resource and waits for the resource to process such action (i.e., the result obtained from the resource in a blocking mode);
5. the scheduling aspect searches for any pre and post conditions for such action execution, and changes the action execution control flow appropriately;
6. once the resource has executed all the preconditions, the action itself and all the postconditions, a result is returned to the agent role, that in turn passes it to the agent that has previously requested the action execution.

### 3.4 Application Example

In order to better explain the above concepts, this section provides an application example similar to the database environment of section 2.2.

```
public class DatabaseAction
extends TaggedAction {
    public Object execute()
        throws Exception {
        DBUtils ut = new DBUtils();
        return ut.query("SELECT * FROM PRODUCTS");
    }
}
```

**Figure 10: The DatabaseAction of the example.**

```
public class DBUtils {
    @Precondition(
        className="org.aglets.r4r.examples.DBUtils"
        methodName="init", actionClassName=
            "org.aglets.r4r.examples.DatabaseAction",
        continueIfFails=false)
    @Postcondition(
        className="org.aglets.r4r.examples.DBUtils",
        actionClassName=
            "org.aglets.r4r.examples.DatabaseAction",
        methodName="finish")
    public Vector query(String sql);
    public void init();
    public void finish();
}
```

**Figure 11: The DBUtils environment-dependent library.**

Imagine an environment where there is a DiscoverRole role that owns a DatabaseResource that allows agents to query a database. Such resource exposes a DatabaseAction as shown in Figure 10. The action is very simple: it creates a DBUtils object used to query the database with an SQL query, reporting then the result to the caller code. The DBUtils class represents an environment-dependent library, and it is where the pre and post conditions must be attached to. To keep the example simple, as shown in Figure 11 the database code requires, before the execution of a query (query()) of setting up and tearing down the database connection and related resources (respectively the methods init() and finish()). In order to force the right set-up and tear down of the database resources, the query() method is annotated with a precondition and a postcondition. Please note that the annotation indicates the exact class and method to be executed as pre and post conditions.

Finally, the agent shown in Figure 12 iterates over its own role actions searching through the OperationDescriptors for an operation named PRODUCT\_SEARCH, that is then executed.

It is important to note that the agent simply executes the action through the `execute()` method, without having any knowledge of any change to the action control flow due to the application of conditions by the `Scheduling` aspect. In other words, the run-time execution is transparent to the agent execution.

Conditions can be different from site to site, and can be even not required on certain environments. This is where the value of the role deployment becomes more important: the roles and the resources are stereotypes common to all environments, but their use depends on per-environment policies.

This brief example demonstrates how the R4R approach can help developers specifying and implementing pre and post conditions for environment dependent actions; more complex scenarios are possible, but they are not shown here due to paper length. Please note that the above example shown one of the two ways R4R supports the definition of policy rules, that is *declaring them where the condition code is required* (see section section:aspect); the other way (*declaring them within the action code*) is not shown due to paper length.

```
public class DatabaseAgent
extends Aglet {
    private Set<Role> roles = ...
    public void run(){
        try{
            for( Role role : this.roles )
                for(TaggedAction action :
                    role.getResourceActions() ){
                    OperationDescriptor desc =
                        action.getActionDescriptor();
                    if(desc.getName()=="PRODUCT_SEARCH")
                        ActionResult result =
                            role.execute(action);
                }
        }catch(Exception e){}
    }
}
```

**Figure 12: The DatabaseAgent that exploits the role action.**

## 4. RELATED WORK

This sections presents a few approaches related to R4R. For a complete comparison of main role approaches, please see [4].

R4R is not the only one role approach that exploits AOP, in fact Kendall's proposal aim at covering different phases of the agent-based application development combining concepts of roles [18]. Briefly this approach binds agents to

roles, chosen among those available in a role catalogue, using AOP. This is clearly different from the R4R approach, that does not touch the role assumption and use, rather the role-action execution, and in fact it is possible to exploit R4R over the Kendall's approach.

RoleX approach is a very dynamic and flexible approach [7]. Briefly RoleX allows an agent to dynamically assume a specific role, merging the agent structure (i.e., class definition) with the role one, as the two were developed as a single entity. Even if very flexible, the RoleX approach does not take into account resources and environment policies. This is the main drawback in RoleX that motivated R4R to be implemented on top of it (see section 3).

The Mobile Agent Reactive Tuplespace (MARS) [9] is a coordination environment for mobile agents that does not involve the concept of roles, but exploits a shared tuple space to let agents communicate. This approach is interesting because it allows the definition of *reactions* to tuple management, that can be similar to the postconditions of R4R. However, the absence of roles and the capability to define only reactions and not more general conditions makes MARS a more limited approach than R4R.

An interesting approach that models interactions through roles and actions is RICA and its Java implementation RICA-J [20]. This approach improves the FIPA standard [14] with support for social concepts, but there is no definition of actions tied to a specific environment. Moreover, it must be noted that RICA does not clearly distinguish the agent from the role that it is playing, since the role represents effectively the agent behavior (and not an *enhanced* behavior). This implicitly means that there is no separation of concerns in agent and role development, and reflectively in agent and action exploitation.

Contract4J [13] is an approach not specifically related to bagent systems. The Contract4J aim is to enforce *design by contract* and its implementation has several parts common to that of R4R. In fact, Contract4J exploits annotations and AOP to (statically) enforce contracts between software components. Nevertheless, it must be noted that Contract4J is a static approach exploited at the design phase, while the R4R approach is a dynamic approach exploited at the deployment and execution phase.

## 5. CONCLUSIONS AND FUTURE WORK

This paper has presented the Resources For Roles (R4R) approach, an extension to RoleX that ease the role deployment and its integration in the final environment.

The idea of R4R is to embed resources in roles and to apply environment rules to resource actions, so to ease the integration of roles in the environment itself. Thanks to the capability of expressing pre and post conditions, R4R allows a

concrete role deployment to an execution environment. The conditions will be used by R4R to dynamically change the action execution flow in order to adapt it to the policies of the execution environment.

R4R does not require role developers to know specific environmental details and rules, allowing a high degree of separation of concerns between the role development and the administration of the execution environment itself.

Even if already publicly available, the R4R approach will be improved by future work. In particular the adoption of XML to describe condition will ease the R4R adoption, since it will no more require Java skills to role deployers.

## REFERENCES

- [1] The Aglets Mobile Agent Platform, *Web Site: <http://aglets.sourceforge.net>*
- [2] The AspectJ Project, *Web Site: <http://www.eclipse.org/aspectj/index.php>*
- [3] Cabri G., Ferrari L., Leonardi L., "Role Agent Pattern: a Developer Guideline", *The 2003 IEEE Systems, Man and Cybernetics Conference, RBC, Washington DC, USA, October 2003*
- [4] Cabri G., Ferrari L., Leonardi L., Zambonelli F., "A Survey about Role-based Interaction Proposals for Agents", *Technical report DII-AG-2005-1*
- [5] Cabri G., Ferrari L., Leonardi L., Zambonelli F., "Connecting Ambient Intelligent Components via Dedicated Middleware: an Agent Approach", *Workshop on Ambient Intelligence - Agents for Ubiquitous Environments at AAMAS 2005, Utrecht, The Netherlands, July 2005*
- [6] Cabri G., Ferrari L., Leonardi L., "A Role-based Mobile-Agent Approach to Support E-Democracy", *Applied Soft Computing, Vol. 6, pp.85-99, ISSN 1568-4946, 2005*
- [7] Cabri G., Ferrari L., Leonardi L., "Injecting Roles in Java Agents Through Run-Time Bytecode Manipulation", *IBM Systems Journal, Vol. 44, No. 1, pp.185-208, 2005*
- [8] Cabri G., Ferrari L., Leonardi L., Quitadamo R., "Role Suggestion for Agents by Overhearing", *The International Journal of Intelligent Control And Systems, Vol. 12, No. 2, pp. 179-185, June 2007*
- [9] Cabri G., Leonardi L., Zambonelli F., "MARS: a Programmable Coordination Architecture for Mobile Agents", *IEEE Internet Computing, Vol.4, No.4, pp.26-35, July-August 2000*
- [10] Cabri G., Ferrari L., Leonardi L., Mamei M., Zambonelli F., "Uncoupling Coordination: Tuple-based Models for Mobility", *Chapter of the book Mobile Middleware, Taylor and Francis CRC Press, London-UK, ISBN 0849338336, September 2006*
- [11] G. Cabri, L. Ferrari, L. Leonardi, "Supporting the Development of Multi-Agent Interactions via Roles", *The 6th International Workshop on Agent-Oriented Software Engineering (AOSE) at AAMAS 2005, Utrecht, The Netherlands, July 2005, Lecture Notes in Computer Science N. 3950, pp.154-166*
- [12] Cabri G., Ferrari L., Zambonelli F., "Role-based Approaches for Engineering Interactions in Large-scale Multi-Agent Systems", *Software Engineering for Multi-Agent Systems II, Lecture Notes in Computer Science n. 2940, pp. 243-263, April 2004*
- [13] Contract4J: Design by Contract for Java *Web Site: <http://www.contract4j.org/contract4j>*
- [14] The Foundation for Intelligent Physical Agents (FIPA), *Web Site: [www.fipa.org](http://www.fipa.org)*
- [15] Gregor K., Lamping J., Mendhekar A., Maeda C., Lopes C., Loingtier J. M., Irwin J., "Aspect-Oriented Programming", *Proceedings of the European Conference on Object-Oriented Programming, vol.1241, pp.220242*
- [16] Hirsh B., Fisher M., Ghidini C., "Programming Group Computations", *Proceedings of the First European Workshop on Multi-Agent System (EUMAS), 18-19 December 2003, Oxford, UK*
- [17] Luck M., McBurney P., Preist C., "Agent Technology: Enabling Next Generation Computing A Roadmap for Agent Based Computing", *AgentLink Web Site: <http://www.agentlink.org/roadmap>*
- [18] Kendall E. A., "Role Modelling for Agent Systems Analysis, Design and Implementation", *IEEE Concurrency, Vol.8, No.2, pp.34-41, April-June 2000*
- [19] Prasad M. V., Lesser V. R. , Lander S. E., "Learning Organizational Roles in a Heterogeneous Multi-agent System", *Working Notes for the AAAI Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems, Sandip Sen, pages 72-77, 1996*
- [20] Serrano J. M. , Ossowski S., "On the Impact of Agent Communicative Languages on the Implementation of Agent Systems", *Cooperative Information Agents VIII, Lecture Notes in Artificial Intelligence, ISSN 0302-9743, Springer, 2004*
- [21] Zhu H., Zhou M., "Role-based collaboration and its kernel mechanisms", *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, Volume 36, Issue 4, July 2006 Page(s): 578 - 589*