A Role Model for Description of Agent Behavior and Coordination

Yunus Emre Selçuk[1], Nadia Erdoğan[1]

[1] Istanbul Technical University, Faculty of Electrical and Electronic Engineering, Computer Engineering Department, Maslak, TR-34469, Istanbul, Turkey
selcukyu@itu.edu.tr, erdogan@cs.itu.edu.tr

Abstract. This paper compares the enhanced role model JAWIRO (Java with Roles) with another role model and a design pattern for implementing roles. These three approaches are compared on the basis of their abilities and performances. It is shown that role models are valuable tools for modeling dynamic real world entities as they provide many useful abilities without a significant performance overhead. The dynamic nature of agents represents a good domain for using roles for describing both their behaviors and their coordination. A brief introduction to role models and the capabilities of JAWIRO are also included in this paper.

1 Introduction

Software systems are constantly getting more complex in order to keep up with the ever changing, dynamic and heterogeneous nature of the current real world scenarios. Complex systems become easier to understand when they are described in terms of acts and responsibilities of the elements they contain [1]. Such a description leads to a better separation of concerns and therefore to better modeling. Roles allow agents to dynamically acquire capabilities to perform specific tasks, and therefore enable separation of concerns and code reusability in software development and maintenance [2].

Separation of concerns leads to the separation between the algorithmic issues and the interaction ones [3]. Roles represent a good paradigm for modeling interactions among agents. A role can be built to represent an interface for interactions, providing a set of common instruments for dealing with and allowing interactions among entities. Furthermore, roles help the modularization and the organization of MAS, separating responsibilities and rights among entities involved [4].

Agent oriented techniques are well suited for modeling complex and distributed systems [5]. As the notion of role is frequently applied for conceptualizing the behavior of human individuals, roles can be used for describing the behavior of individual agents in a multiple agent system [6]. MAS implementations such as [3, 7] can be found in literature which use roles in their approaches. Roles are also used for encapsulating the interactions between agents [7, 8]. In ROPE; roles provide a well defined interface between agents and cooperation processes, which enables an agent

to read and follow the normative rules given by the cooperation process even if not known to the agent before [8].

This paper presents a role model implementation, JAWIRO, which enhances Java with role support for better modeling of dynamically evolving real world systems. JAWIRO provides all expected requirements of roles, as well as providing additional functionalities without a performance overhead when executing methods. An example application is also described in order to demonstrate how roles can be used in MAS.

2 Related Work

The BRAIN framework [7] covers the development of agent-based systems while modeling agent interactions with roles. The RoleX extension [3] introduces an interaction infrastructure for the BRAIN framework for Java mobile agents using bytecode manipulation for role operations. Although bytecode manipulation proves itself to be useful, it can breach the Java security mechanism. Therefore, bytecode manipulation is not used in JAWIRO.

An agent can be thought as a role or a set of roles [6]. However, having agents as roles is somewhat controversial. A role is defined as a class that defines a normative behavioral repertoire of an agent in [9]. We believe that roles are useful for representing both the coordination and the responsibilities of agents as they provide a good separation of concerns.

3 The Role Concept from the Role Models' Viewpoint

The role concept comes from the theoretical definition where it is the part of a play that is played by an actor on stage. Roles are different types of behavior that different types of entities can perform. Kristensen [10] defines a role as follows: A role of an object is a set of properties which are important for an object to be able to behave in a certain way expected by a set of other objects.

A *role model* specifies a style of designing and implementing roles. As such, coding by using roles can be called as *role based programming* (RBP). RBP provides a direct and general way to separate internal and external behaviors of objects. When a role model is built in an object oriented environment, RBP extends the concepts of OOP naturally and elegantly.

Object oriented programming is based on specialization at the class level, e.g. *(class level) inheritance*. However, specialization at the instance level is a better approach than specialization at the class level when evolving entities are to be modeled. In this case, an entity is represented by multiple objects, each executing a different role that the real-world entity is required to perform. In role based programming, an object evolves by acquiring new roles and this type of specialization at the instance level is called *object level inheritance*. When multiple objects are involved, the fact that all these objects represent the same entity is lost in the regular OOP paradigm unless the programmer takes extra precaution to keep that information

such as utilizing a member in each class for labeling purposes. Role models take this burden from the programmer and provide a mechanism for object level inheritance.

Object level inheritance successfully models the *IsPartOf* [11] relation where class level inheritance elegantly models the *IsA* [11] relation. As both types of relationship are required when modeling of real world systems, both types of inheritance should coexist in an object-oriented environment. Therefore, many role models are implemented by extending an object-oriented language of choice, such as INADA [12], DEC-JAVA [13], the works of Schrefl and Thalhammer [14] and Lee and Bae [15], etc.

4 Overview of JAWIRO

Our role model JAWIRO extends the Java programming language with role support. Java has been chosen as the base language because even though it has advanced capabilities that help to its widespread use, it lacks features to design and implement roles in order to model dynamic object behaviors. JAWIRO implements all basic features of roles as well as additional capabilities that can be expected from roles, e.g. the extended features of roles.

4.1 Features of Roles

Definition of the basic features of roles varies slightly among different researchers [10, 14]. We believe the basic features of a role model should contain the following:

- Roles can be gained and abandoned dynamically and independently of each other.
- Roles can be organized in various hierarchical relationships. A role can play other roles, too.
- The notion that a real world object is defined by all its roles is preserved, e.g. each role object is aware of its owner and the root of the hierarchy.
- An entity can switch between its roles any time it wishes. This means that any of the roles of an object can be accessed from a reference to any other role.
- A role can access member variables and methods of other roles by means of the two previously described features.
- Class level inheritance can be used together with object level inheritance.
- Entities can be queried whether they are currently playing a certain type of role or a particular role object.
- An entity can have more than one instance of the same role type. Such roles are called *aggregate roles* and are distinguished from each other with an identifier.
- Different roles are allowed to have member variables and methods with same names without conflicts.

In addition to the above listed basic features, JAWIRO implements the following extended features as well:

- Roles can be suspended and then resumed.
- A role can be transferred to another owner without dropping its sub roles.
- Multiple object level inheritance is supported.

- Any public member variable or method of any participant of a role hierarchy can be accessed solely by its name, without a direct reference to its owner. In case of identical names, the most evolved member is returned.
- Previously mentioned behavior can be overridden by setting dominant nodes in a role hierarchy.
- Both consultation and delegation mechanisms are supported.
- Abnormal role bindings are prevented.
- Persistence is supported, so that users are able to save entire role hierarchies to secondary storage devices for later use.

More details and usage examples of the basic features of roles can be found in [16] and the extended features are explained in detail in [17] except for persistency, the latest and final enhancement of JAWIRO.

Kendall is one of the researchers who pointed out some useful properties of roles that can be used in MAS [18]. In compliance with Kendall's statements, the role model of JAWIRO does not exist to replace class models. On the contrary, JAWIRO extends the strongly typed and class based nature of Java with the basic and extended features of roles. Roles are implemented as first class objects so that they can be instantiated, generalized, specialized and aggregated; just as Kendall stated [18].

Another compliance of JAWIRO with Kendall's statements [18] is the dynamic nature of JAWIRO. Role hierarchies can be evolved by means of gaining, transferring, suspending, resuming and resigning roles. The ability of JAWIRO to access members of a participant of a role hierarchy without referencing that particular participant, combined with the dominance ability, ensures this evolution. Roles in JAWIRO can constrain each other with the use of constraint managers [17], again as mentioned in [18].

4.2 Role Model of JAWIRO

JAWIRO models relational hierarchies of roles with a tree representation. Such hierarchical representation enables better modeling of role ownership relations. This leads to easier and more robust implementation of roles' basic and extended characteristics.

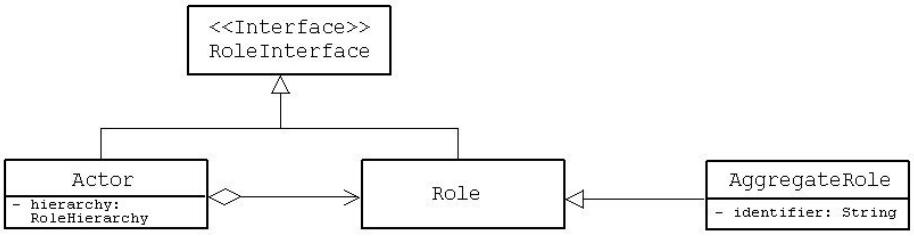

Fig. 1. The UML schema of JAWIRO API

The UML schema of JAWIRO API is given in Fig. 1. The `Actor` class models the real world objects which can be the root of a role hierarchy. The `Role` class models the role objects. The `Actor` and `Role` classes implement the `RoleInterface` as these two classes share some characteristics of roles. The aggregate roles are implemented by deriving a namesake class via class-level inheritance from `Role` class. The backbone of the role model is implemented in the `RoleHierarchy` class, where each `Actor` object has one member of this type.

5 Evaluating JAWIRO

This section evaluates JAWIRO in terms of both its features and performance. Schrefl and Thalhammer's role model [14] for Java and an implementation of a design pattern for roles, *role relationship* [19], are used for comparison. The role relationship pattern is extended from the role object pattern [19].

Schrefl and Thalhammer's role model [14] is implemented in Java and is available for academic use. This role model is based on Gottlob, Schrefl and Rock's previous work [20] in Smalltalk. Schrefl's recent work with Thalhammer [14] supports all primary features of roles.

5.1 Feature Comparison

JAWIRO is currently, to the best of our knowledge, the only role model which supports both the basic and extended features of roles. Table 1 shows some key features of the compared approaches, as well as other recent role models for Java. To the best of our knowledge, there are no properties that are supported by other models but not by JAWIRO. Benefits of the unique features of JAWIRO can be found in [17].

Table 1. Feature based comparison of recent role models in Java

	Jawiro	Schrefl et al. [14]	Role Rel. Pattern [19]	Dec-Java [12]	Lee& Bae [15]
Base language	Java	Java	Java	Java	Java
Aggregate roles	+	+	+	+	–
Hierarchy support	+	+	–	+	–
Run-time role checking	+	+	+	–	–
Preventing role binding anomalies	+	–	–	–	+
Object level multiple inheritance	+	–	–	–	–
Member/method access without referring its owner.	+	–	–	–	–
Dominant roles	+	–	–	–	–
Delegation and consultation support	+	–	–	–	–
Role transfers	+	–	–	–	–
Suspending and resuming roles	+	–	–	–	–
Persistency	+	–	–	–	–
Role searching optimization	+	–	–	–	–

We have kept the implementation of the role relationship pattern [19] as its original. This pattern can be extended to support additional features of roles, but we think this would cause us loose focus on JAWIRO. Another important work, Schrefl and Thalhammer's role model [14], supports all basic features of roles. However, it does not support the extended features of roles. This role model is available for download as a JAR file, together with its documentation.

5.2 Performance Comparison

The objective of the performance comparison is twofold. Firstly, we need to compare JAWIRO's performance with that of another role model. Secondly, we need to determine whether a significant overhead is introduced or not when roles are incorporated in an application.

The benchmarking code first creates a role hierarchy with a given depth and degree. The tree representing the hierarchy is a balanced one. However, the role relationship pattern does not support hierarchies of depth greater than two. In this case, the code creates an equal number of role objects but adds all of them to the same object, the root. The benchmarking code then executes commands representing the basic features of roles.

In order to see how changes in the size of a role hierarchy affect performance, we should be able to create hierarchies with arbitrary depth and degree. This need leads to arbitrary number of role objects as well. Even trees with small values of depth and degree can lead to thousands of role objects. It is practically impossible to create such great numbers of different role classes. Therefore, we've used *aggregate roles,* as defined in Section 3.1 among the basic features of roles and named as *qualified roles* in Schrefl's model [14], as role objects in the benchmark. The results of our benchmarks are given in Table 2. They are obtained by using an Intel platform with 2.8GHz Pentium 4 CPU, i865 chipset, 512MB RAM and JDK 1.5.0_03.

Table 2. Benchmark results in Intel platform. Hierarchy depth is 6 and its degree is 3

	Average exec. time (msec.)					
	Without Optimization			With Optimization		
Stages	Jawiro	Schrefl	Pattern	Jawiro	Schrefl	Pattern
Create members	0,009	N/A	0,009	0,0044	N/A	0,000
Add roles	0,009	N/A	0,000	0,004	N/A	0,000
Construct hierarchy	0,017	0,082	0,009	0,009	0,073	0,000
Role checking	0,018	0,009	0,041	0,0002	0,009	0,041
Role switching	0,018	0,030	0,041	0,0002	0,030	0,041
Role execution	0,006	0,005	0,004	0,003	0,003	0,002
Switching execution	0,043	0,047	0,056	0,004	0,033	0,044
Checking switching execution	0,068	0,092	0,110	0,006	0,065	0,087

There is a slight difference in creating role hierarchies between JAWIRO and Schrefl's model [14]. In JAWIRO, role objects are instantiated with an arbitrary

constructor and added to an owner any time the programmer wishes by issuing the `RoleInterface.addRole(Role)` call. These two calls represent the first and the second stages given in Table 2. On the other hand, role objects must be bound with an owner during instantiation when using Schrefl's model. This represents the third stage given in Table 2. For easier comparison, the third stage for JAWIRO is calculated by adding the execution times of stages 1 and 2 in Table 2. We will name the first three stages as *building phase* and the others as *running phase*.

JAWIRO uses an optimization mechanism which will be explained shortly. For now, consider the un-optimized results given in Table 2 first. These results show that using role models instead of a pattern introduces two or three-fold overhead during the building phase. However, that overhead diminishes as the running phase is more important than the building phase. The building phase is run for only once at the beginning of the code but the operations in the running phase will be repeated continuously during the lifetime of the application program. Table 2 also shows that role models are always faster in the running phase and JAWIRO is usually faster than Schrefl's model. It is also seen that the overhead of role execution is virtually zero in all role based approaches.

One of the unique features of JAWIRO is an optimization mechanism for searching and switching roles. It is a wise choice in dynamic and persistent systems such as JAWIRO to search for existence of a role before switching to that role. Whenever the existence of a role is checked, JAWIRO keeps this particular role in a private member. When handling a following role switching command, JAWIRO first checks that private member. If the requested role is the one kept in the private member, that role is returned without searching the role hierarchy. Subsequent switching requests to that same role also return the role kept in the private member. The optimized results given in Table 2 show that this mechanism proves itself to be useful. Execution times of the benchmark stages are either halved or shortened tenfold when commands are rearranged in an order that makes use of the optimization mechanism.

The same benchmark code gives different results in AMD platforms. In a PC with Athlon XP 2500+ CPU, nForce2 chipset, 512MB RAM and JDK 1.5.0_03; performance of Schrefl's model [14] becomes 25% better and performance of the role relationship pattern [18] becomes 40% better while JAWIRO's performance becomes 20% poorer.

In order to investigate how changes in the size of a role hierarchy affect performance, the benchmarks for the running phase are repeated for trees with different depth values on the Intel platform. The results obtained for JAWIRO are presented in Table 4.

Table 4 shows that JAWIRO causes no overhead when executing roles, regardless of the size of the role hierarchy. Even if there are one million roles in the hierarchy, the biggest overhead that JAWIRO introduces is as small as one tenth of a millisecond. Table 5 shows the same benchmark in the Intel platform, using Schrefl's model and the role relationship pattern.

Table 4. Effects of hierarchy size on JAWIRO, using Intel platform. n represents number of the role objects in the hierarchy

Degree=3 (constant)	n=39; n^2=1,521	n=120; n^2=14,400	n=363; n^2=131,769	n=1,092; n^2=1,192,464
Depth	**4**	**5**	**6**	**7**
Role checking (n^2 operations)	0,003	0,012	0,018	0,058
Role switching (n^2 operations)	0,003	0,003	0,018	0,055
Role execution (n operations)	0,000	0,000	0,001	0,001
Switching execution (n^2 operations)	0,000	0,011	0,018	0,055
Checking switching execution (n^2 ops.)	0,005	0,003	0,020	0,055

Table 5 shows that the results for Schrefl's model and the role relationship pattern are similar with the results for JAWIRO, with one exception. Schrefl's role model uses hash tables to keep track of roles, therefore its performance for role checking operations are unaffected with the growing sizes of the role hierarchies.

Table 5. Effects of hierarchy size on Schrefl's role model and the role relationship pattern, using Intel platform

Degree=3 (constant)	Schrefl's model				Role relationship pattern			
Depth	**4**	**5**	**6**	**7**	**4**	**5**	**6**	**7**
Role checking	0,008	0,008	0,010	0,011	0,009	0,014	0,041	0,131
Role switching	0,014	0,017	0,030	0,073	0,003	0,017	0,041	0,130
Role execution	0,000	0,000	0,000	0,000	0,000	0,000	0,001	0,002
Switching execution	0,013	0,017	0,031	0,074	0,008	0,016	0,044	0,135
Checking switching execution	0,022	0,033	0,061	0,147	0,006	0,031	0,088	0,273

6 Describing Behavior with Roles in a Simple Team Application

This section demonstrates how roles can be employed for a team of agents by using JAWIRO. Consider a multiplayer shooter game where the team members are computer controlled entities, i.e. bots. There can be mobile bots representing soldiers, as well as immobile turrets and medical stations. Figure 2 shows a role hierarchy for modeling such an environment.

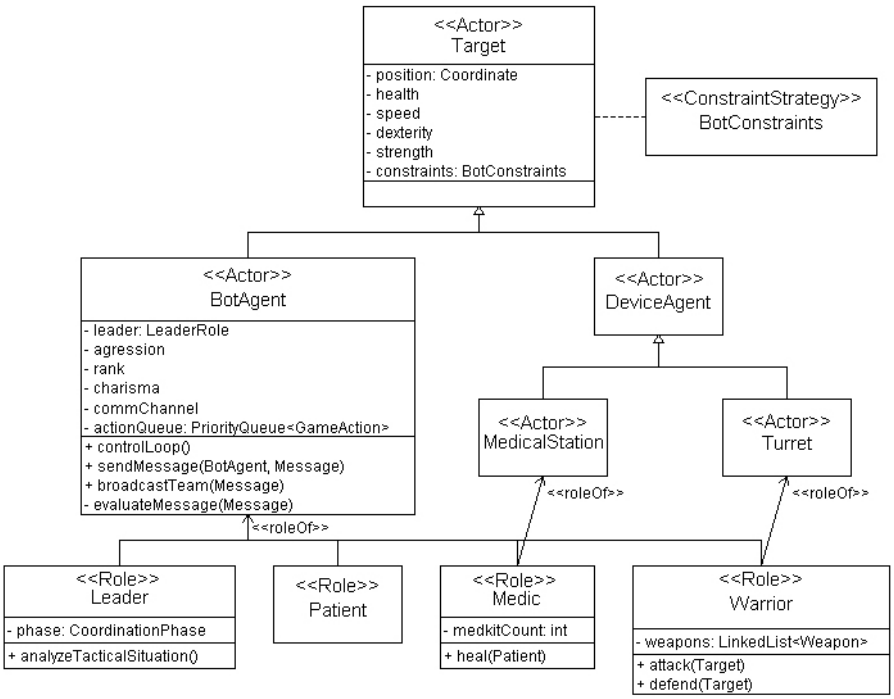

Fig. 2. A role hierarchy for modeling agents in a shooter game

All agent types are created from the `Target` class which contains the common properties of different bot types. The `Target` class extends the `jawiro.Actor` class; so that a `Target` instance can be root of a role hierarchy. Mobile bots are instances of the `BotAgent` class, which can play all available roles. The `Leader` role can be played by only one member of a team, while there can be multiple bots playing the `Warrior` or the `Medic` role. Moreover, a `BotAgent` instance in need of medical treatment can play the `Patient` role.

There are also immobile bots in the application domain, i.e. medical stations which heal nearby patients and turrets which attack the enemies in its range. These devices are modeled with namesake classes. The `Medic` role can be reused for the medical stations; therefore a `MedicalStation` instance can play the `Medic` role but it cannot become a `Warrior`. Similarly, a `Turret` instance can play the `Warrior` role but it cannot become a `Medic`.

Let's examine some possible situations where the characteristics of roles can be put into good use. A warrior who noticed that he is moderately injured broadcasts a call for a medic and it gains the `Patient` role. If his health is further dropped, he can become crippled. In such case, his `Warrior` role is suspended until he is attended by a bot playing the `Medic` role. However, he can heal himself if he plays a `Medic` role, too.

The behavior of a bot is determined by its current roles. The control loop of each bot checks the owned roles and determines which actions to be taken. The specific

commands of the leader are executed in topmost priority, as long as the necessary roles exist. For example, a bot attacks to or defends a target with an appropriate weapon if it has the `Warrior` role. If there are no specific orders, bots make decisions which fit the current situation. For example, a bot having the `Medic` role looks for nearby allies to heal.

The example application domain imposes some rules on role binding operations. These rules can be enforced by the constraint managers of JAWIRO, given in detail in [13]. Briefly; constraint managers are called before a role is gained, lost, suspended or resumed so that they have a chance to allow or disallow the operation. The `BotConstraints` class contains the rules of the restricted role bindings previously described. Moreover, it can automate actions such as resuming the `Warrior` role after loosing the `Patient` role if the soldier was previously crippled.

Roles can be used for representing and/or implementing the coordination of multiple agents as well. Just as roles representing individual behavior can be added to individual agents; roles representing cooperation and coordination rules can be added to agents, too. The way how the individual agents cooperate can be altered according to the current state of the environment by defining the roles modeling the coordination rules necessary at that instant as dominant to the rest.

7 Results

Role models provide an abstraction that can unify diverse aspects of an agent system such as collaboration protocols and task models. Additionally; agents, objects, and people can all play roles, so that a role model can span multiple layers in a software system [18]. The results of our assessment show that role models are valuable tools in modeling dynamically evolving systems. Role models introduce no overhead when executing roles and the overhead introduced in the running phase is quite insignificant. As patterns and ad-hoc approaches do not support all features of roles, they can be only used where they cover all required abilities. When a project needs the basic features of roles, role models become the obvious choice. Any role model can be chosen at this stage, considering the platform that the software is targeted. However, JAWIRO is the only role model that supports all of the extended features of roles as well and presents a runtime performance well enough for non real-time systems. The JAWIRO role package, long with its API documentation and usage examples, is available in http://www.yunusemreselcuk.com/jawiro/index.html. Currently, we are working on tailoring JAWIRO for agent based systems to be used in describing both the responsibilities and the coordination of agents.

References

1. Pacheco, O., Carmo, J. : A Role Based Model for the Normative Specification of Organized Collective Agency and Agents Interaction. Autonomous Agents and Multi-Agent Systems 6 (2003) 145–184

2. Cabri, G., Ferrari, L., Leonardi, L. : Applying security policies through agent roles: A JAAS based approach. Science of Computer Programming (Article in Press)
3. Cabri, G., Ferrari, L., Leonardi, L. : Exploiting runtime bytecode manipulation to add roles to Java agents. Science of Computer Programming. 54 (2005) 73–98
4. Zhu, H. : A Role Agent Model for Collaborative Systems. Proc. Int'l Conf. on Information and Knowledge Engineering, (2003)
5. Jennings, N.R. : An agent-based approach for building complex software systems. Communications of the ACM 44/4 (2001) 35–41
6. Odell, J.J., Parunak, H.V.D., Brueckner, S., Sauter, J. : Temporal Aspects of Dynamic Role Assignment. Proc. 4th Int'l Workshop on Agent-Oriented Software Engineering. (2003) 201–213
7. Cabri, G., Leonardi, L., Zambonelli, F. : BRAIN: a Framework for Flexible Role-based Interactions in Multiagent Systems. Proc. Conf. On Cooperative Information Systems. (2003)
8. Becht, M., Gurzkil, T., Klarmann, J., Muscholl, M. : ROPE: Role Oriented Programming Environment for Multiagent Systems. Fourth IECIS Int'l Conf. on Cooperative Information Systems. (1999) 325–333
9. Odell, J.J., Parunak, H.V.D., Fleischer, M. : The Role of Roles in Designing Effective Agent Organizations. In Software Engineering for Large-Scale Multi-Agent Systems, Springer-Verlag (2003)
10. Kristensen, B.B. : Conceptual Abstraction Theory and Practical Language Issues. Theory and Practice of Object Systems 2/3 (1996)
11. Zendler, A.M.: Foundation of the Taxonomic Object System. Information and Software Technology 40 (1998) 475–492
12. Aritsugi, M., Makinouchi, A. : Multiple-Type Objects in an Enhanced C++ Persistent Programming Language. Software - Practice and Experience. 30/2 (2000) 151–174
13. Bettini, L., Capecchi, S., Venneri, B. : Extending Java to Dynamic Object Behaviours. Electronic Notes in Theoretical Computer Science. 82/8 (2003)
14. Schrefl, M., Thalhammer, T. : Using roles in Java. Software - Practice and Experience. 34 (2004) 449–464
15. Lee, J-S., Bae, D-H. : An Enhanced Role Model for Alleviating the Role-Binding Anomaly. Software - Practice and Experience. 32 (2002) 1317–1344
16. Selçuk, Y.E., Erdoğan, N. : JAWIRO: Enhancing Java with Roles. Proc. 19th Int'l Symposium on Computer and Information Sciences. (2004) 927–934
17. Selçuk, Y.E., Erdoğan, N. : JAWIRO: An Extended Role Model For Java. Proc. Int'l Conference on Computational Intelligence. (2004) 207-210.
18. Kendall, E.A. : Role Models – Patterns of Agent System Analysis and Design. BT Technology Journal. 17/4 (1999) 46–57
19. Fowler, M. : Dealing with Roles. Unpublished paper. http://martinfowler.com/apsupp/roles.pdf
20. Gottlob, G., Schrefl, M., Röck, B. : Extending object-oriented systems with roles. ACM Trans. on Information Systems. 14/3 (1996) 268–296