

The Adaptive Agent Model: Software Adaptivity through Dynamic Agents and XML-based Business Rules

Liang Xiao and Des Greer

School of Computer Science, Queen's University Belfast, Belfast, BT7 1NN, UK

email: { l.xiao, des.greer }@qub.ac.uk

Abstract

The Adaptive Agent Model (AAM) achieves adaptivity in three areas. i) Agents are associated with business rules. These rules are configured by business analysts in a natural language style, using a Business Rules Editor, stored in an XML repository and executed by agents. ii) The vocabularies of rules are easily changed. A Metadata Editor is described for this purpose, allowing business analysts to dynamically create new objects/attributes or edit existing ones during the operation of the software system. Once new rules involving new vocabularies are defined and assigned to agents, they will take effect immediately at run-time, reflecting new policies and ontologies available to the system. iii) The collaboration between agents can be modified. We describe an Architecture Editor which facilitates control of the agent interaction. Adaptivity in these three areas, using the associated editors, is demonstrated. The approach ensures that agents implement up-to-date requirements, reflecting desired current behaviour, without the need for frequent system rebuilds. AAM is illustrated and evaluated using of an e-business example.

1. Introduction

Software evolution and maintenance is expensive and accounts for the majority of software lifecycle costs [1]. To make systems more *adaptable*, effort has been concentrated on making them easily changed by engineers. Better still, would be the ability to make systems *adaptive*, where systems change their behaviour according to their context [2].

In the next section we discuss various approaches towards making systems adaptive. Section 3 describes related work and the motivation of using agents and business rules for adaptivity. Section 4 starts with a case study selected to illustrate the need for better adaptivity and to provide a vehicle for later discussions. Our Adaptive Agent Model (AAM) is then described in detail and Section 5 provides an evaluation of the work.

2. Current Approaches to Adaptivity

In this section we will look at some sample approaches to achieving adaptability and adaptivity. One example is the use of the *Strategy Design Pattern*. The

concept of design patterns has been around for some time now, the best known patterns having been documented in [3]. Some patterns such as the 'Strategy' pattern describe ways of adapting behaviour without the need to rewrite. Briefly, the required algorithms for a class are initially predicted and written. Later, during program operation these can be selected from, as appropriate. Hence, limited adaptivity is achieved, if and only if the new behaviour has been predicted.

Another prominent approach has been the use of *Coordination Contracts*. Assuming in a component-based system each component provides stable functionality, by externalising the coordination between them as contracts [4] (text-based descriptions), dynamic reconfiguration can be achieved. System evolution consists of adding and removing contracts without modification of the computation implemented in the participating components. Thus, flexible combination of components brings more variable behaviour than with the Strategy pattern. However, components will still have to be re-developed even if the functionalities are only slightly changed.

The *Adaptive Object Model (AOM)* is described in [5] as a framework that models business units with metadata, which will be interpreted at run-time. This is achieved using the 'TypeObject', 'Property' and 'Strategy' patterns. By using the TypeObject pattern, unpredicted classes can be created at run-time from generic ones, in the same way as objects are instantiated from classes. As in the Strategy pattern, AOM also allows dynamic behaviour to be configured for classes. Different rules can be associated with objects, and they can be constructed at runtime to represent the required behaviour. As with the Strategy pattern, behaviour can be adapted. In addition, its business concepts are extensible with the TypeObject/Property pattern. Nonetheless, the AOM has several weaknesses. Firstly, since classes are created dynamically in the AOM, if they are to be persistent, their definition must be stored in a database. The hard-coded original classes, therefore, do not represent business abstractions because most information about the business is in the database. Hence, developers may find the architectures required in this approach hard to understand and maintain. Secondly, AOM in common with using the Strategy pattern can only customise behaviour within the limits of what has been predicted. Thirdly, there is no easily accessible central model so that analysts and clients alike may find it difficult to track the current state of the system.

Considering all these sample approaches and others, the primary motivations for this work are to reduce the cost of maintenance and, related to this, the time required to carry out maintenance [1]. One way to achieve this is to empower users to take responsibility for evolving software. This reduces the delay and cost in introducing change. Ultimately (and somewhat idealistically), the cheapest option is to have the software self-adaptive. One fundamental property of all these existing approaches is that they are OO-based. Since objects are passive and traditionally have fixed methods, either it is impossible or inconvenient to make them adaptive. In addition, it is likely to bring side effects (like AOM does). Our goal is to find a method that breaks this tradition and moves the responsibility for making change from the developer to the user and in so doing reduces cost.

3. Agents and Business Rules for Adaptivity

The object-oriented paradigm facilitates design by use of such principles as modularity and information hiding, but this implies ease of redesign rather than adaptation during operation. Intelligent/autonomous agents have been proved to be useful for bringing dynamics, flexibility and adaptivity to travel planning [6], coordinated product development and manufacture [7] and manufacturing systems control [8].

Software agents are defined as follows: “An agent is an encapsulated computer system that is situated in some environment, and that is capable of flexible, autonomous action in that environment in order to meet its design objectives” [9]. JADE [10] (Java Agent DEvelopment) framework is one of the platforms that conforms to FIPA [11] standards for developing agents. In JADE, agents are able to carry out several concurrent tasks in response to different external events. Sending and receiving messages are the two main activities of agents.

Our hypothesis is that a system consisting of agents, where the behaviour can be configured dynamically as policies or rules will provide better adaptivity. Firstly, unlike standard objects, agents are active. Instead of using static methods which are to be invoked and have the same effects all the time, agents are granted the flexibility to choose how to react. Secondly, because it is not possible to know what customers want in advance, requirements that are embedded in code are difficult to change. For that reason we propose an approach that uses rules with a XML format to store requirements, the adaptation of which is made by business people by using a set of easy-to-use editors. While agents are running, their behaviour adapts immediately according to the new business requirements from business people. Agents understand, translate and execute them at run-time, reflected in their continuously changing behaviour according to the change from the editors, thus achieving easier adaptivity.

There has been previous work [12] [13] on modelling business rules in agent systems. However these are not easy to implement and their rules are not configurable at runtime by business people.

4. Solution Approach - Adaptive Agent Model

Our AAM approach aims at giving business people the direct control over their systems and their operations using a set of editors/tools to adapt systems at runtime. Prior to describing these editors, we will introduce an illustrative case study.

4.1. Case Study

To illustrate the need for adaptivity and to use in our discussion later, we introduce a small ecommerce case study. Suppose a retailer has an online shop to serve university staff and students. The retailer has certain policies that provide various discounts to students including special offers to particular groups, departments or individuals during changing seasons. Customers can become “premium” customers, if their spending exceeds a certain amount. Discount policies will change periodically and new criteria may arise for these. For example, one scenario is that credit could be used to evaluate external buyers and additional policies on credit would be made to provide buyers with better discounts. The retailer may not predict that new concepts such as “credit” will be introduced in the future. Further, possible changes in the collaboration between the retailer and suppliers may occur, such as changing suppliers, if one cannot meet the demand.

In the case study above we can identify three categories of changes: new business policies; new business concepts; and new business architectures / collaborations. These categories can be managed dynamically using three corresponding editors, which combine to implement the AAM.

4.2. Adapting Business Policies

In AAM, business policies are described in terms of business rules. A business rule is a compact statement about some aspect of a business. It is a constraint in the sense that a business rule lays down what must or must not be the case [14]. Often, business rules are hard-coded into programs, but keeping business rules distinct from code has many advantages, including the fact that they remain highly understandable and accessible to non-programmers.

We distinguish business rules into two categories: *global rules* that represent business policies applicable to the whole business domain and *local rules* applicable to individual agents. Global rules handling is discussed in this section and local rules in Section 4.4.

Global rules have very basic IF-THEN structures which state that if an entity of a system has a certain value or a certain relationship with another entity, then a certain action is to be performed such as assigning a value to an entity. To make the global rules accessible by business analysts, a web-based *Business Rules Editor* has been developed. Figure 1 shows part of its interface. The business entities which are used to compose these rules are abstracted and listed on the interface for simple selection, so that those people who have no programming experience can specify their relationships as rules in a straightforward way. Business rules are stored in a central XML document and, whenever rules are updated the document is updated accordingly. The rules list is retrieved from the document and shown on the interface for viewing and editing.

Business Rule in Editing:

IF : customer name == Liang Xiao

THEN : customer discount = 0.15

Priority : 2, Rule Id : 1

Update this rule Save it as a new rule

Stored Business Rule:

- IF customer.name == Liang Xiao THEN customer.discount = 0.15 Priority: 2
- IF customer.occupation == researcher THEN customer.discount = 0.10 Priority: 3
- IF customer.type == premium THEN customer.discount = 0.05 Priority: 5

Figure 1: Business Rules Editor with three rules definition and the first one being edited

At the moment, we allow a simple template for rules taking the form:

*IF objectName1.attributeName1 op1 value1
THEN objectName2.attributeName2 op2 value2
Priority value3*

op1 and op2 correspond to “less than”, “equal to” or “greater than”. The higher value3 is, the lower the priority of the rule.

Figure 2 shows a sample business rule.

*IF customer.type = premium
THEN customer.discount := 5% Priority: 5*

Figure 2: A sample business rule

A template rule structure modelled in XML is shown in Figure 3. Ultimately, value1/value2 will be replaced with other business entities and more complex and hierarchical logic structures will be used to specify rules.

```
- <rule>
  <id>ruleId</id>
  <condition>
    objectName1.attributeName1 op1 value1
  </condition>
  <action>
    objectName2.attributeName2 op2 value2
  </action>
  <priority>value3</priority>
</rule>
```

Figure 3: A business rule template stored as XML

We use a Business Rules Manager Agent (BRMA) to oversee the communication between agents and the application of rules. The actual communication between the customer agent and the BRMA to apply the rule given in Figure 2 is described in an Agent Communication Diagram (ACD), shown in Figure 4.

The ACD is an extension to UML and is inspired by the Agent-Object-Relationship model in [12]. It shows agents, rules and their interactions, via messages.

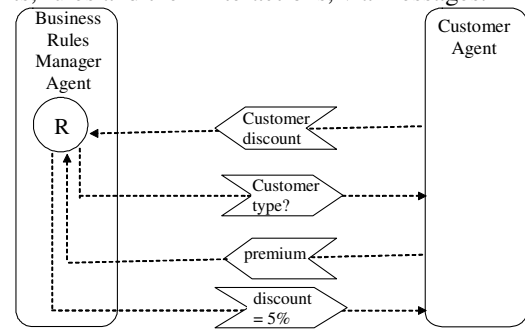


Figure 4: An Agent Communication Diagram describing an enquiry for a business policy

The BRMA determines the relevant rules and ranks them using the <priority> tag. A message is then sent to the initialising agent asking about the context information it holds according to the <condition> tag. If the returned result fulfils the condition, then the action in <action> is executed, otherwise it moves to the next rule.

In the above scenario, the customer agent obtains from the BRMA the discount value for the customer it represents. In the beginning, the customer agent finds and sends the BRMA an enquiry about customer discount. When the request is received, the BRMA then searches its stored global rules set and generates a list of rules relevant to customer discount. The above rule is found and its action is checked for relevance. In this case, it is an assignment statement for customer discount, and so is applicable. If the rule has the highest priority, the BRMA executes it. Thus, the BRMA needs to determine if the type of the customer is premium. Since it does not have customer information itself, it sends a message to the customer agent requesting this information. On determination of this fact, the BRMA can finally apply the action and sends the discount value to the customer agent. In generic form, the steps the BRMA takes to answer enquiries from other agents are:

1. Get a list of relevant rules according to the <action> tag.
2. Get the rule that currently has the highest priority according to the <priority> tag.
3. Send messages to the initialising agent asking about context information it holds according to the <condition> tag.
4. According to the returned result, if the condition is satisfied, then reply to the initialising agent with the appropriate value set in the rule. Otherwise, remove this rule from the rule set and go to Step 2. A default value is returned, if this is the last rule in the rule set.

The messages sent by the BRMA are generated dynamically. In our example, at any time, the attributes relevant to the discount can be specified through the Business Rules Editor (In fact, the next section will demonstrate how the attribute definitions are also dynamic). Agents will use new or changed rules,

immediately they are available. Referring to the rule defined in Figure 2, suppose we add another two rules so creating the following larger rule set (shown in Figure 1)

- i) IF *customer.name = Liang Xiao*
THEN *customer.discount := 0.15* Priority: 2
- ii) IF *customer.occupation = researcher*
THEN *customer.discount := 0.10* Priority: 3
- iii) IF *customer.type = premium*
THEN *customer.discount := 0.05* Priority: 5

The BRMA will order them: rule (i), rule (ii), and rule (iii). Assume a researcher “Liang Xiao” is a premium customer, the associated agent checks the discount for him. Rule (ii) and rule (iii) will not apply but rule (i) will be executed. For another researcher “Des Greer”, rule (ii) will apply.

Rules are independent entities and can be structured as a hierarchy. Rules with highest priorities are the most specific ones, while rules with lower priorities are likely to have more generic applicability. Thus, rules overriding can be achieved as required to enable specific behaviour.

4.3. Adapting Business Concepts

Business concepts refer to the terms allowed in business rules. In our approach, a *Metadata Editor* allows new objects to be declared via a web interface, along with new attributes. New objects and attributes created are immediately available and can be accessed via the Business Rules Editor. Therefore, new business behaviour involving these new concepts can be created, reflecting new requirements, as agents execute the new rules defined on them.

For example, Figure 5 shows the Metadata Editor web interface being used to add a new attribute, “credit” to the object “customer”.

Figure 5: Adding a new attribute “credit” to the “customer” object through Metadata Editor

The updated XML document is as shown in Figure 6 with an additional entry registered with the “customer”.

```
- <object>
  <name>customer</name>
  - <attributes>
    <attribute>name</attribute>
    <attribute>type</attribute>
    <!-- ... more attributes ... -->
    <attribute>credit</attribute>
  </attributes>
  <behaviour/>
</object>
```

Figure 6: Updated XML metadata

This makes it possible to define the following new rule:

IF *customer.credit > 5000*
THEN *customer.discount := 0.20* Priority: 1

via the Business Rules Editor as shown in Figure 7.

Figure 7: New rule with new attribute “credit”

Without changes to the rules already defined and without restarting the agent system, the new attribute is registered and a new rule using it has been added to the business rules document, with immediate effect. Customer “Liang Xiao” will enjoy a discount of 20% instead of 15%, if he has a credit higher than 5000, because this rule is applicable and has the highest priority, provided all rules defined in the previous section are not changed.

4.4. Adapting Collaboration

In agent systems business processes are implemented by the collaboration of agents. The management of this collaboration requires the agent architecture to be modelled and in AAM the model should be adaptive. Architecture in software systems describes the communication structures for system entities. In object-oriented systems, objects are aware of which other objects they will pass messages to but are unaware of which objects will pass messages to them. Full architecture independence requires that the detail of where objects will send messages should also be hidden [15]. In our approach, there will be a series of message passing among coordination agents to accomplish every business task. We therefore introduce an *Architecture Editor* as a modelling tool that enables the specification of the agent collaborations and message flow control. The Architecture Editor will allow the creation of *local rules*. These rules control the directions that agents receive and send messages. On receipt of any message, an agent reads its local rules, analyses them and finds out the appropriate agents to send messages to. When the definitions change, rules change and agents get the changed rules to execute. A sample screen from the editor is shown in Figure 8 with the definitions of local business rules given on the left of the window.

With the use of the editor, this approach achieves two-way encapsulation by using dynamic local business rules that are formed in XML structures. The agent behaviour is guided by rules so that they do not need to know who they will contact in advance. Hence the collaborations between them are dynamic and encapsulated in rules.

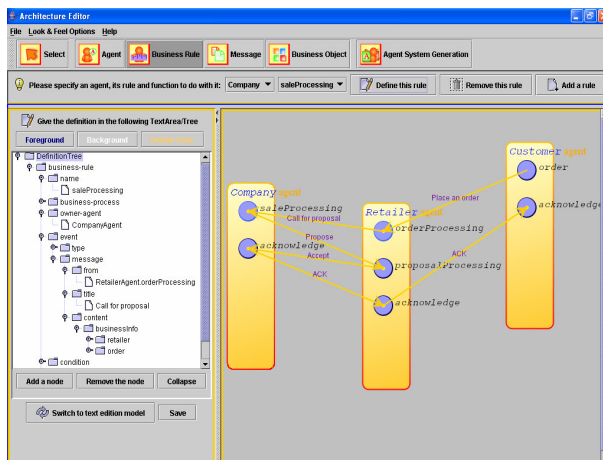


Figure 8: Architecture Editor with the requirements information of agent interaction models, rule reaction patterns and message flows captured

The Agent Communication Diagram (ACD) is used to help the design of multi-agent behaviour and it is the basis of the collaboration modelling approach. It shows in Figure 8 the Architecture Editor is used for an ACD specification, describing a process where a customer orders products from a business company through a retailer. The representation of rule “saleProcessing” for “Company” agent in XML is shown in Figure 9.

```

- <local-rule>
  <name>saleProcessing</name>
  <business-process>
    retailer business
  </business-process>
  <owner-agent>Company</owner-agent>
  - <event>
    <type>receipt of message</type>
    - <message>
      <from>Retailer.orderProcessing</from>
      <to>Company.saleProcessing</to>
      <title>Call for proposal</title>
      - <content>
        - <businessInfo>
          - <retailer> ... </retailer>
          - <order>
            <id>10010001</id>
            - <product>
              <classification>
                book
              </classification>
            </product>
            </order>
          </businessInfo>
        </content>
      </message>
    </event>
    <condition>
      order.isOrderAttractive() = true
    </condition>
    - <action>
      <type>send a message</type>
      - <message>
        <from>Company.saleProcessing</from>
        <to>Retailer.proposalProcessing</to>
        <title>Propose</title>
        - <content>
          - <proposal>
            <id>10011101</id>
            <businessInfo> ... </businessInfo>
          </proposal>
        </content>
      </message>
    </action>
    <priority>5</priority>
  </local-rule>

```

Figure 9: Sample definition of the local rule “saleProcessing” owned by “Company” agent

Rule definitions encoded in XML include details about the causing trigger event, the data content of the message being received and a series of (condition, action, priority) triplets. Whenever an event occurs to an agent, usually on receipt of a message, an agent parses its local rule set and chooses the appropriate rule to deal with the event. With the Architecture Editor, XML-based rules can be generated in their <event> and <action> sections when incoming/outgoing messages are specified and <condition> and <priority> sections are given afterwards.

“Company” agent reacts to the receipt of a message by executing this rule in the following way:

- On the event of receiving an incoming “Call for proposal” message from the “Retailer” agent,
- If the condition of the rule is satisfied, that is, isOrderAttractive () is true, then perform an action, that is, generate a business proposal for the order, including price for the order, order dispatch time, etc. and send these as a “Propose” message to the “Retailer” agent,
- Update its beliefs, that the customer placing the order is interested in the products listed in the order.

In the case of “saleProcessing” in Figures 8&9, suppose all previous rules managed by “Company” agent are not applicable. The agent parses the message just received and finds out that the message has come from an agent with the name “Retailer”. Hence, it matches with the event described in “saleProcessing” because the content of the <event>/<message>/<from> tells it that the rule deals with messages from the retailer agent. By executing business logics defined on the business object of “order” it knows whether the order is attractive or not. The “order” object can be built from the content of the <businessInfo>/<order> structure of the received message. If the order is attractive then the condition for executing the rule is satisfied and another business object of “proposal” is generated with the assistance of business logics. A corresponding message will then be structured and sent to the “Retailer”, again as specified in <action>/<message>/<to>. Finally the “Company” agent adds the knowledge that this retailer agent has an interest in the books which it just ordered to its own beliefs. In the process if the event does not match or the condition is not satisfied, the next candidate rule with the highest priority will be tested for applicability and executed in the same way.

The whole process is dynamic so that business processes can be specified and updated as required through the editor and XML definitions for agent rules. On detection of any event, agents get the most up-to-date rules set relevant to them and react in the way specified, just at that moment. This allows adjustment of agent communication structure dynamically and therefore the architecture of the system.

As in the other two editors, the Architecture Editor also uses the business rules document as the database.

Once business processes are specified graphically in the editor, agent interaction models, rule reaction patterns and message flows are established by the editor accordingly. Agent systems will be automatically generated. The agent behaviour is governed by rules so that we achieve a model driven communication architecture in the generated system. Thus agents in the system can have their partners changed in order to accomplish various business tasks. This is achieved simply by changing the related rules of the affected agent. Not only the architecture and the agent behaviour, but also the details of message contents are adaptive.

5. Evaluation and Conclusion

In our agent system, agent knowledge is modelled as business rules. Agents retrieve and update their knowledge from the business rules document and behave accordingly. There are three aspects of such knowledge where, in each case, adaptivity is achieved:

- i) Addition of new business concepts (via the Metadata Editor),
- ii) Adjustment of business policies (via the Business Rules Editor) and
- iii) Change of business architectures (via the Architecture Editor).

Table one considers the three levels of adaptivity that we have identified and summarises how well AAM provides these in comparison with other techniques. None of the other techniques provide full adaptivity at all three levels. Moreover, each of the other techniques requires expert programming knowledge, unlike AAM which is aimed at business analysts, making use of UML-like diagrams and natural language rules.

Table One: A comparison of existing techniques with AAM in providing adaptivity

<i>Approach Area of Adaptivity</i>	<i>Strategy Design Pattern</i>	<i>Coordination Contract</i>	<i>AOM</i>	<i>AAM</i>
Business Concepts	No	No	Yes	Yes
Business Policies	Yes, but only if all future policies can be predicted	Yes, built only if new policies can be expressed as contracts.	Yes, but only if all future policies can be predicted	Yes, additional text-based rules suffice
Business Architectures	No	Yes	Yes	Yes

The Adaptive Agent Model provides adaptivity, in the sense that the editors define the environment where the agents operate and the agents adapt to this context. The next step will be to exploit further the adaptive nature of agents. Currently in AAM the content and priority of rules is decided by the user. Although this may seem intuitive, it would be desirable to automate trivial and repetitive tasks. For example, a set of rules may be automatically executed in a certain order to accomplish a user demand according to their previous behaviour in similar circumstances. Also, where a rule

fails, alternative rules may be automatically chosen. Using agent beliefs may help to achieve this and ultimately, agents could set rule priorities themselves and choose the most appropriate rules for whatever situation occurs.

The Adaptive Agent Model will be made more powerful and more flexible, but work so far indicates that it will contribute in a novel and substantive way to the business need for adaptivity in systems.

References

- [1] Manna, M., "Maintenance Burden Begging for a Remedy", *Datamation*, 53-63, 1993.
- [2] Lieberherr, K., "Workshop on Adaptable and Adaptive Software", *Proceedings of the Tenth Conference on Object Oriented Programming Systems Languages and Applications*, 149-154, 1995.
- [3] Gamma, E., Richard, H., Johnson, R. & Vlissides, J., "Design Patterns", Addison-Wesley, 1994.
- [4] Andrade, L. & Fiadeiro, J.L., "An Architectural Approach to Auto-Adaptive Systems", *22nd International Conference on Distributed Computing Systems Workshops*, 2002.
- [5] Yoder, J.W. & Johnson, R., "The Adaptive Object-Model Architectural Style", *Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture: System Design, Development and Maintenance*, 3-27, 2002.
- [6] Yim, S.H., Ahn, J.H., Kim, W.J. & Park, J.S., "Agent-based adaptive travel planning system in peak seasons", *Expert Systems with Applications*, 27(2) 211-222, 2004.
- [7] Jia, Z.H., Ong, K.S., Fuh, J.Y.H., Zhang, F.Y. & Nee, A.Y.C., "An adaptive and upgradable agent-based system for coordinated product development and manufacture", *Robotics and Computer-Integrated Manufacturing*, 20(2) 79-90, 2004.
- [8] Goh, W.T. & Zhang, Z., "An intelligent and adaptive modelling and configuration approach to manufacturing systems control", *Journal of Materials Processing Technology*, 139(1-3) 103-109, 2003.
- [9] Jennings, N.R., "On Agent-Based Software Engineering", *Artificial Intelligence*, 14(2) 145-189, 2000.
- [10] JADE platform, <http://sharon.cselt.it/projects/jade/>.
- [11] Foundation for Intelligent Physical Agents, <http://www.fipa.org/>.
- [12] Wagner, G., "The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior", *Information Systems* 28(5), 2003.
- [13] Laleci, G.B., Kabak, Y., Dogac, A., Cingil, I., Kirbas, S., Yildiz, A., Sinir, S., Ozdakis, O. & Ozturk, O., "A Platform for Agent Behavior Design and Multi Agent Orchestration", *Agent-Oriented Software Engineering Workshop, the Third International Joint Conference on Autonomous Agents & Multi-Agent Systems*, 2004.
- [14] Morgan, T., "Business Rules and Information Systems", Addison-Wesley, 2002.
- [15] Object Management Group, Inc., "Applying UML 2 to Model-Driven Architecture", 250 First Ave. Suite 100, Needham, MA 02494, USA, 2003.