Modeling, Auto-generation and Adaptation of Multi-Agent Systems

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
phone: +44 (0)28 9097 4656
fax: +44 (0)28 9097 5666

Abstract. We propose a lightweight approach that provides mechanisms for dynamic agent behavior at run-time. Agent collaborations are modeled in UML diagrams and agent behaviors are encoded in XML-based business rules. The combination of these captures the behavioral requirements and governs inter-agent and intra-agent behaviors. A CASE tool has been developed to enable the dynamic specification of agent behaviors and the generation of the agent systems. Agents get the appropriate rules in XML format and then translate and execute them at run-time. These rules are externalized and so maintenance effort is reduced, since there is no need to recode and regenerate the agent system. Rather, the system model is easily configured by users and agents will always get up-to-date rules to execute at run-time. The approach is illustrated with the aid of an e-business example and its efficacy discussed.
Keywords. adaptivity, agent, business rule, e-Business, requirements, UML

1 Introduction

Agent-oriented systems differ from object-oriented systems in that agents are active, while objects are passive. Thus, agents have the goal of having dynamic behaviors. Therefore, agent systems should be easily adaptable, being easily changed by engineers. Better still, would be that they were adaptive, where systems change their behavior according to their context [1].

Although many tools and techniques are available for agent-oriented systems development, there is no unified and mature way to do it. What is more, existing agent platforms, like JADE [2], require designers and developers to code agent behaviors in fixed class methods and the way to write them varies one platform to another. This lack of uniformity of approach means that maintaining agent systems is potentially expensive. Being able to automatically generate agent systems and adapt their behaviors with changing requirements would alleviate this maintenance burden. We describe the Adaptive Agent Model as an approach that generates agent systems from models and configures agent behaviors and ontologies at run-time. AAM continually

reflects new requirements immediately in the generated agent systems. AAM assumes the use of UML and stores business rules in XML format.

1.1 Agent systems and platforms

Various agent platforms are available, the JADE [2] (Java Agent DEvelopment) framework being one of them. JADE is aimed at developing multi-agent systems and applications conforming to FIPA [3] standards. With JADE, an agent is able to carry out several concurrent tasks in response to different external events. Sending and receiving messages are the two main activities of agents. Traditionally, developers are required to write code for every agent, their behaviors during communication, message passing, and ontology used in programming languages before building and implementing agents running on agent platforms.

1.2 Business rules and agent behaviors

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 [4]. 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. XML-based rules have been used in the IBM San Francisco Framework [5] as templates to specify the contents and structures for code that is to be generated. With this approach, changing of XML rule templates allows mappings to new object structures. Figure 1 shows an example, where a generic XML rule has been converted to a specific Java method, getDiscount () in this case.

```
<Rule>
  <Target> Attributes </Target>
  <Condition> scope = public </Condition>
  public &type; get&u.name;() {
    return iv&u.name;;
  }
</Rule>
```

If the name of one of the public attributes for an **Order** class was “discount”, and its type “Double”, then this template would generate:

```
public Double getDiscount() {
  return ivDiscount;
}
```

Fig. 1. Example of code generation using rules

Because agent behaviors represent actual system requirements and are subject to change, the application of business rules to the agent world should offer similar advantages as in the object world.

1.3 Agent-oriented UML

Agent-Object-Relationship (AOR) [6] models show social interaction processes in organizational information systems in the form of interaction pattern diagrams. These model agents, ordinary objects, events, actions, claims, commitments, and reaction rules which dictate behaviors. AOR can be viewed as an extension of UML for agent systems and is capable of capturing the semantics of business domains. The construction and editing of rules are not in its scope. Moreover, how agents, objects and rules work together are not described adequately. However, it provides a good notation system for the agent world and we later adapt and use it for our conceptual modeling of agents, rules and their interactions.

2 Background and motivation

Current approaches to agent-oriented system design and implementation are fundamentally based on the identification of agent interaction protocols, message routing, and the precise specification of the ontology. This need for complete upfront design makes it difficult to manage agent conversations flexibly and to reuse agent behavior subclasses [7]. Using Agent Patterns [8] is one way for better code encapsulation and reuse. It is argued in [8] that much research work such as Gaia [9], MaSE [10], and Tropos [11] emphasize only the design of basic elements like goals, communications, roles, and so on. Although the reuse of patterns, which are observed as recurring agent tasks appearing in similar agent communications, can reduce repetitive code, the chance that a pattern can be reused without change is low. Reuse of patterns in different context is not straightforward. In addition, this approach is not adaptive since system requirements change means that models need to be changed, patterns need to be re-written and agent classes re-generated.

State machines have also been suggested for agent behavior modeling [12] and the Extensible Agent Behavior Specification Language (XABSL) has been specified [13] to replace native programming language and to support Behavior Modules design. Intermediate code can be generated from XABSL documents and an agent engine has been developed to execute this code. The language is good at specifying individual agent behavior, but cannot express behaviors that involve inter-agent collaboration. Moreover, although agent behaviors are modeled in XABSL, they must be compiled before being executed by the agent engine. Thus, changing the XABSL document always requires re-compilation.

Agent behaviors are modeled as workflow processes in [14] and a Behavior Type Design Tool is described for constructing behaviors. This approach provides a convenient way to compose agent behaviors visually. However, its use of Agent Behavior Representation Language (ABRL) to describe agent interaction scenarios and “guard expressions” to control the behavior execution order does not facilitate the modeling of systems as a whole. Further the approach does not offer an agent system generation solution.

In answer to the weaknesses of the existing approaches, we propose the use of UML diagrams to model agent interactions and XML-based business rules to encode agent behaviors. Stable business classes are available for these rules to act upon. Rules govern agent behaviors, make decisions for agents in various contexts, have controls over the invocation of business classes and are adaptive. Agents and their rules are specified using an AAM-CASE tool. Rule definitions in terms of inter-agent behaviors are generated from the given UML diagrams, while rule definitions in terms of intra-agent behaviors are specified in the tool, so that different business classes can be used in different message processing or agent actions. Agent systems can be generated from the tool. Each agent reacts to events according to the XML-based rules document at run-time. Rules can be changed in the tool very easily and it makes use of extensible ontologies.

3 Solution approach: Adaptive Agent Model

We emphasize the integration of UML diagrams which model inter-agent relationships and XML rule definitions each of which describes an individual agent behavior. UML model information will become part of the XML definitions and enable agents to understand their communication with the outside world.

3.1 Case study

To illustrate our approach and to use in our discussion later, we introduce an ecommerce case study. Suppose a retailer runs an online shop. The retailer has an association with customers and also with various supplier companies, who may or may not serve the retailer, depending on different conditions. Overall, the relationship between customers, retailers and supplier companies can change at any time. The business vocabulary is also changeable and the decision making process for each company, retailer and customer is unpredictable.

3.2 Structural model: Agent Diagram

Structural models are built through *Agent Diagrams,* and show agents, business rules, business classes and their relationship. Agents manage rules and rules manage the invocation of business classes. Such models are used for agent identification, agent relationship identification, and eventually building an Agent/Rule/Class hierarchy. They are later the basis for the behavioral models.

Agents are identified to represent distinct conceptual domains. Agent Diagram has Class Diagram, the backbone of UML [15] as its counterpart in the object-oriented models. In AAM agents are regarded as superior to classes. Each rounded cornered box represents an agent and is divided into three compartments. The top compartment holds the name of the agent, the middle compartment holds the classes managed by the agent along with their instantiation and the bottom compartment holds the rules that

govern the functions of the agent. They resemble a class name, an attribute list, and an operation list constituting a class diagram in the OO world.

Agent systems always require interactions among many agents. Such interactions are modeled as message passing between agents. The message sender requests a service from the message receiver. The message receiver uses its internal business objects for the computation required to fulfill the request and then, possibly, takes a further action. Different situations will arise and these are modeled as rules that agents should obey. Thus, a rule is responsible for the behavior of an agent in dealing with a particular situation. Multiple rules can be defined to let the agent collaborate with other agents to achieve different goals.

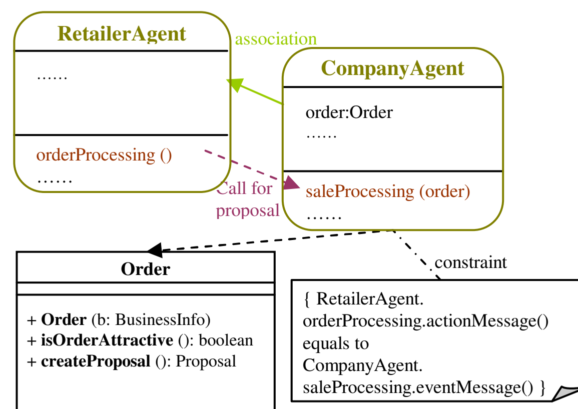

Fig. 2. The Agent Diagram for case study

In Figure 2, “RetailerAgent” and “CompanyAgent” are the two identified agents for our case study. “RetailerAgent” has a rule “orderProcessing” that will construct an object with type “BusinessInfo”, package it into a “Call for proposal” message and send the message to “CompanyAgent”. To respond to such requests, “CompanyAgent” will offer a deal, if the order is attractive, using the rule “saleProcessing”. Thus, we have an association relationship between the two agents involved and a constraint for them. They resemble an association between two classes and a constraint for classes in the OO world. During the processing of rule “saleProcessing”, an “Order” object will be constructed from the received “BusinessInfo” structure and the constructed object should pass an “isOrderAttractive” check before “CompanyAgent” proceeds to offer a deal, “Proposal” for the order. Thus, such a business class of “Order” is managed by “CompanyAgent” and it has at least three methods that will be invoked by the agent rule.

3.3 Behavioral model: Agent Communication Diagram

Agent Diagrams capture the static relationship between different entities and depict the whole system. *Agent Communication Diagrams* are used to model the interaction

of agents. Such behavioral models organize agents, rules and messages around business processes. For every business process, all participating agents will appear in the diagram, with message passing between them to accomplish certain business goals.

Software Architecture refers to the communication structures for system entities. In traditional 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 [16]. In agent-oriented systems, business processes are implemented by the collaboration of agents. The management of this collaboration requires the agent architecture to be well modeled. In order to generate agent systems and be able to adapt them afterwards without re-generation and re-compilation, full architecture independence (two-way encapsulation) is required, and the interaction information should not be hard-coded so that agents can adapt their collaboration in communication according to changing requirements, two techniques are used in combination for this purpose.

In our approach, UML is used to model agent collaborations, describing how message passing among coordination agents can accomplish business tasks. UML diagrams provide a blueprint for involved business rules, the composition elements of our diagrams. Each rule governs an individual agent behavior in participating collaborations. Rules are connected to form a flow of decision making, process by process, one decision being made at each connection point. As such, the model visualizes the actual system function in a sequence of agent actions dictated by rules. User specified agent collaborations in UML diagrams are used to generate the inter-agent part of the rules definition, in XML format. It is through these rules that agent systems are adapted both in collaborations and internally without re-code or re-generation, since we let agents get appropriate rules to execute only at run-time, and rules get configured continuously through supporting tools we provide.

Accurately, the UML diagram used for the design of multi-agent behaviors is the Agent Communication Diagram. It has been developed based on Agent Diagram and used for the generation of agent systems. Figure 3 describes the process from the case study, where a customer orders products from a business company through a retailer. Business classes are not shown on the diagram but the invocations of their methods are, such as, the one for condition check. R2 has been shown previously as “saleProcessing” in the bottom compartment of “CompanyAgent” in Figure 2.

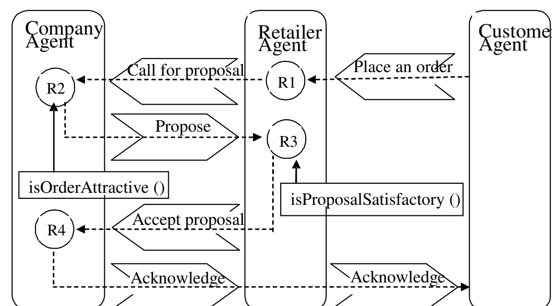

Fig. 3. An Agent Communication Diagram describing a business process

```
– <business-rule>
    <name>saleProcessing</name>
    <business-process>retailer business</business-process>
    <owner-agent>CompanyAgent</owner-agent>
  – <global-variable>
    – <var>
        <name>order</name>
        <type>Order</type>
      </var>
    – <var>
        <name>proposal</name>
        <type>Proposal</type>
      </var>
    </global-variable>
  – <event>
      <type>receipt of message</type>
    – <message>
        <from>RetailerAgent.orderProcessing</from>
        <to>CompanyAgent.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>
    <processing>
      order = new Order (businessInfo)
      proposal = order.createProposal ()
    </processing>
    <condition>
      order.isOrderAttractive() == true
    </condition>
  – <action>
      <type>send a message</type>
    – <message>
        <from>CompanyAgent.saleProcessing</from>
        <to>RetailerAgent.proposalProcessing</to>
        <title>Propose</title>
      – <content>
        – <proposal>
            <id>10011101</id>
            <businessInfo> … </businessInfo>
          </proposal>
        </content>
      </message>
    </action>
    <priority>5</priority>
  </business-rule>
```

Fig. 4. The XML definition for rule “saleProcessing” owned by “CompanyAgent”

We encode diagrammatic rules in the models with a main structure of {event, processing, condition, action, priority} in XML. On receipt of an event (normally modeled as a message), an agent would act on it if the condition of the rule which is defined to deal with this event for the agent is satisfied. Rules are considered according to their priorities set by users. The XML representation for rule R2 is given in Figure 4.

UML diagrams are good at showing collaborations among agents, while XML rules as such are good at precise definition of agent behaviors: this is what the UML Diagrams lack [15]. In the diagram of Figure 3, “CompanyAgent” reacts to the “Call for proposal” message from “RetailerAgent” by executing the above specifically defined rule “saleProcessing” in XML.

3.4 AAM-CASE tool

A tool has been developed to enable the specification of the agent collaborations, rule definitions and message flow control. Figure 5 captures a window from this tool showing the construction of an Agent Communication Diagram in its main panel. Rules can be defined either in XML text or using a more user-friendly tree structure as shown in the left panel. By using this tool, part of the <event> and <action> sections of rules can be generated when incoming/outgoing messages are specified and, <processing>, <condition> and <priority> sections are given afterwards by users. XML code is eventually generated from the completed tree structure and saved in a rules document.

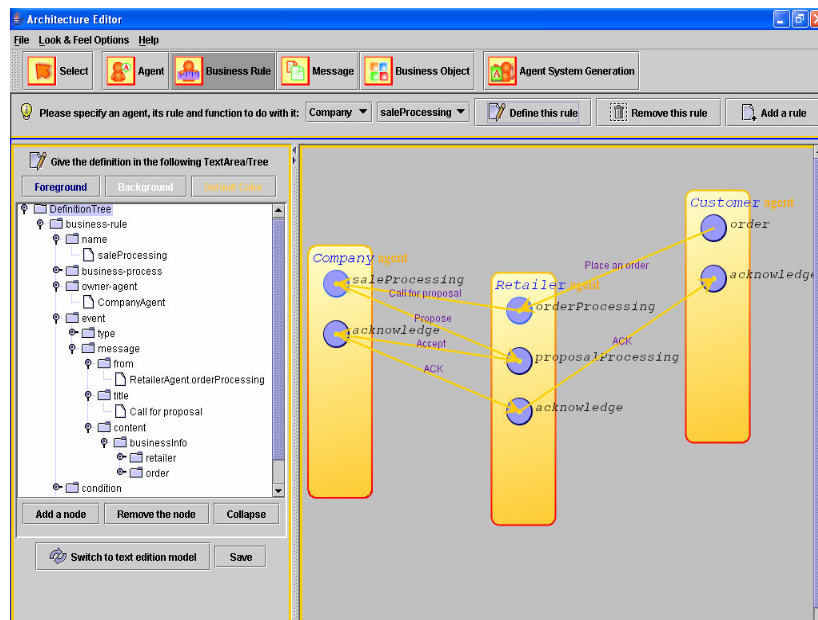

Fig. 5. AAM-CASE tool

3.5 Agent system implementation and deployment

The AAM-CASE tool uses a business rules document as the database. Once business processes are specified graphically in the tool, agent interaction models, rule reaction patterns and message flows are established accordingly. Agent systems are automatically generated such that each rule maps to an agent behavior. Program code is not generated at this moment. Instead, XML-based rules are plugged in and are subsequently translated by agents at run-time. While the system is running, rules can be updated through the tool, so that agent behaviors are continuously updated. The system runs on the JADE platform and can be in a distributed network. All agents access the central XML-based rules document via a parsing package. This allows dynamic adjustment of agent communication structure and therefore the architecture of the system. Sample pseudo code for agent behaviors is shown in Figure 6.

A shared module called “Rule” is used by all behaviors with its ability to access the XML definition of rules and assemble corresponding objects. The methods getPriority(), getEvent(), and getAction() are provided by “Rule”.

```
thisAgent. addBehavior (Rule thisRule) {
    thisBehavior. setPriority (thisRule. getPriority ());
    Order order;
    Proposal proposal;
    Message m = thisAgent. receiveMessage ();
    while (m != null)
    {
       Agent fromAgent = m. getSenderAgent ();
       if (fromAgent. equals (thisRule. getEvent (). getMessage (). getFromAgent ()))
       {
          /* the rule is applicable to the received message */
          BusinessInfo businessInfo = (BusinessInfo) m. getContentObject ();
          order = new Order (businessInfo);
          if (order. isOrderAttractive ())
          {
             /* the condition of the rule is satisfied */
             proposal = order. createProposal ();
             Message m2 = new Message ();
             m2. setContentObject (proposal);
             Agent toAgent = thisRule. getAction (). getMessage (). getToAgent ();
             m2. addReceiverAgent (toAgent);
             thisAgent. send (m2);
             /* update this agent's beliefs */
             thisAgent. addBelief (System. getCurrentTime (), fromAgent, m);
          }
       }
      m = thisAgent. receiveMessage ();
    }
}
```

Fig. 6. Pseudo code for one behavior of “CompanyAgent”, mapping to its “saleProcessing” rule

3.6 Adapting inter-agent collaborations

This approach achieves two-way encapsulation. Agent behaviors are guided by rules so that they do not need to know who they will contact in advance. To reflect business process change, the agent behavioral models can be changed easily with the tool. These changes are automatically reflected in the XML definitions for corresponding agent rules, for example, in their <event>/<message>/<from> and <action>/<message>/<to> sections. This enables agents in the running system have their partners changed in order to accomplish the updated business processes. On receipt of any message, an agent reads the most recent rules, analyzes them and finds out the appropriate agents to send messages to. In the case study, we may wish to re-configure the rule "saleProcessing", and let the "CompanyAgent" take a new action in a condition previously not predicted.

Suppose now we wish to introduce a new occasion where if the current "CompanyAgent" does not evaluate the received order request to be "attractive" or can not fulfill the order request, it forwards the order to another "CompanyAgent". This new requirement can be specified, implemented and deployed by agents automatically, via configuring the Agent Communication Diagrams through the tool. The achievement of this dynamic collaboration is through painless model adjustment rather than expensive code change. Further, we achieve a model-driven communication architecture.

3.7 Adapting intra-agent behaviors

The behavior of agents in processing the event, checking the condition, and taking the action is externalized in business rules. This means that they can be configured dynamically. In fact, by changing the <event>, <processing>, <condition>, and <action> fields in appropriate rules, alternative methods of the managed business objects can be selected for invocation. In the case study, we can re-configure the rule "saleProcessing" to invoke a new evaluation method of the "Order" class or event a method of a new "Order" class to check the attractiveness of the order. In addition, we can configure two couplets of <condition> and <action>, so that for ordinary customers and company customers, different ways to generate sale proposals can be used. All this can be carried out at run-time.

3.8 Adapting ontologies

Only business concepts registered through the tool and saved in the rules document may appear in agent messages. When a new business concept is required, it can be registered with its properties, and a new business class with attributes will be generated by the tool. New vocabularies thus can become available for the specification of agent rules through the tree structure on the left panel of the tool (Figure 5). Also at run-time new classes with new methods thus can become available for invocation by the running agent system. Eventually, all agents will be able to understand the new vocabularies the other agents in the system are using even those

registered after the system has been running for a while. Hence, ontologies are always updatable. For the case study, suppose that an additional attribute of the "BusinessInfo" business object is required and added while the system is running, the updated class becomes available to all agents immediately.

4 Evaluation and conclusion

Agent behaviors are modeled and externalized as rules, and represented in UML diagrams. They are centrally managed and easy to be changed through the models or the XML-based definitions. Agent behaviors reflect functional requirements. Because rules are easy to edit, deploying new requirements requires minimal effort. The rules are, in effect, executable requirements.

One weakness of the Adaptive Agent Model is that the framework's externalization of agent behaviors in XML-based rules will degrade the performance of such systems. Every time an agent acts and reacts to events, it will read the rules document, test rules' applicability, find the one with the highest priority, and execute it. Therefore, there is a trade-off between ease of adaptation and performance. However, adding/upgrading hardware or using parallel computation will compensate the cost.

In the future, we expect to achieve self-adaptivity in the AAM where, as agents interact with end users they perceive their behaviors and preferences. As shown in Figure 7, this allows agents to update their beliefs, and so deduce rules that can be added to the central rules document. These inferred rules can be shared and executed by all agents and are subject to amendment. After some time, a mature and reliable rule set, independent of those acquired through the tool can be established.

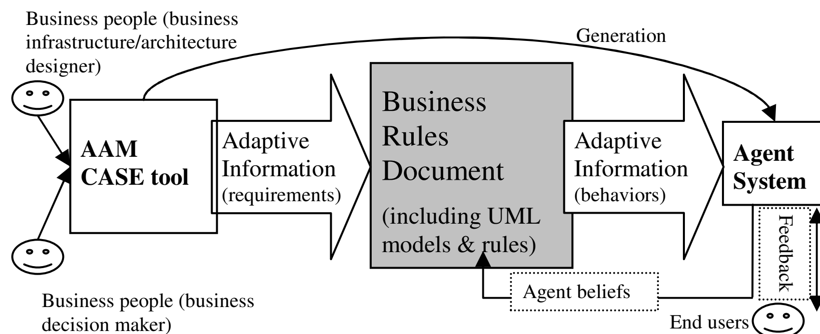

Fig. 7. Future Adaptive Agent Model

AAM would be useful for those domains that have frequently changing requirements where re-development would otherwise be costly. Particularly, AAM should work well when there is collaboration between many different entities and where this collaboration may be subject to adjustment, as a result of changing business processes. AAM is also suitable where the business environment is frequently changing with emerging concepts and behaviors.

Other future work will include the development of richer business rules. 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.

5 References

1. Lieberherr, K., "Workshop on Adaptable and Adaptive Software", Proceedings of the Tenth Conference on Object Oriented Programming Systems Languages and Applications, 149-154, 1995.
2. JADE platform, http://sharon.cselt.it/projects/jade/.
3. Foundation for Intelligent Physical Agents, http://www.fipa.org/.
4. Morgan, T., "Business Rules and Information Systems", Addison-Wesley, 2002.
5. Bohrer, K.A., "Architecture of the San Francisco Frameworks", IBM Systems Journal, 37(2) 156-169, 1998.
6. Wagner, G., "The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior", Information Systems, 28(5) 475-504, 2003.
7. Griss, M., Fonseca, S., Cowan, D. & Kessler, R., "SmartAgent: Extending the JADE Agent Behavior Model", Proceedings of SEMAS, 2002.
8. Cossentino, M., Burrafato, P., Lombardo, S. & Sabatucci, L. "Introducing Pattern Reuse in the Design of Multi-Agent Systems", AITA'02 workshop at NODe02, 2002.
9. Wooldridge, M., Jennings, N.R. & Kinny, D., "The Gaia Methodology for Agent-Oriented Analysis and Design", Journal of Autonomous Agents and Multi-Agent Systems, 3(3) 285-312, 2000.
10. DeLoach, S.A., Wood, M.F. & Sparkman, C.H., "Multiagent Systems Engineering", International Journal on Software Engineering and Knowledge Engineering, 11(3) 231-258, 2001.
11. Castro, J., Kolp, M. & Mylopoulos, J., "Towards Requirements-Driven Information Systems Engineering: The Tropos Project", Information Systems, Elsevier, Amsterdam, The Netherlands, 2002.
12. Arai, T. & Stolzenburg, F., "Multiagent systems specification by UML statecharts aiming at intelligent manufacturing", Proceedings of the First International Joint Conference on Autonomous Agents & Multi-Agent Systems, 11-18, 2002.
13. Lotzsch, M., Bach, J., Burkhard, H.-D. & Jungel, M., "Designing Agent Behavior with the Extensible Agent Behavior Specification Language XABSL", RoboCup 2003: Robot Soccer World Cup VII, volume 3020 of Lecture Notes in Artificial Intelligence, 114-124. Springer, 2004.
14. Laleci, G. B., Kabak, Y., Dogac, A., Cingil, I., Kirbas, S., Yildiz, A., Sinir, S., Ozdikis, 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.
15. Fowler, M., "UML distilled" third edition, Addison-Wesley, 2004.
16. Object Management Group, Inc., "Applying UML 2 to Model-Driven Architecture", 250 First Ave. Suite 100, Needham, MA 02494, USA, 2003.