

Modelling Agent Knowledge with Business Rules

L. Xiao and D. Greer
*School of Computer Science,
Queen's University Belfast
Belfast, BT7 1NN, UK
email: { l.xiao, des.greer }@qub.ac.uk*

Abstract

Multi-agent systems have become increasingly mature, but their appearance does not make the traditional OO approach obsolete. On the contrary, OO methodologies can benefit from the principles and tools designed for agent systems. The Agent/Rule/Class (ARC) framework is proposed as an approach that builds agents upon traditional OO system components and makes use of business rules to dictate agent behaviour with the aid of OO components. By modelling agent knowledge in business rules, the proposed paradigm provides a straightforward means to develop agent-oriented systems based on the existing object-oriented systems and offers features that are otherwise difficult to achieve in the original OO systems. The main outcome of using ARC is the achievement of adaptivity. The framework is supported by a tool that ensures agents implement up-to-date requirements from business people, reflecting desired current behaviour, without the need for frequent system rebuilds. ARC is illustrated with an e-business example.

1. Introduction

Traditionally, human knowledge is transferred into software systems in the form of requirements documents, design models and eventually implemented code, the performance of which precisely reflects the desired behaviour in the required system. The initially captured knowledge is usually documented in UML diagrams. However, the UML models rapidly lose their value as, in practice changes are often done at the code level only.

The idea of emphasising system knowledge, and making them reusable models that can be converted to executable software, is promoted by the up and coming Model Driven Architecture (MDA) [1] [2]. In MDA, models are central rather than an overhead in the development process. MDA proposes a Platform Independent Model (PIM), a highly abstracted model, independent of any implementation technology. This is translated to one or more Platform Specific Models (PSM). Code is finally translated from PSM. In the whole process of PIM→PSM→code, system knowledge is reused and transformed into different forms.

Adopting the objectives of MDA, we propose an agent-oriented paradigm, where knowledge on agent behaviour is structured as business rules. The amendment of these is carried out directly by business people, reflecting desired business requirements. The execution of these is performed by agents, so reflecting deployed requirements. Such an approach offers several advantages. Firstly, traditional OO methodology is the basis of the framework, no radical development model change is involved and, therefore, minimum effort is needed for developing such systems. Secondly, knowledge of the system can be more easily maintained, as business rules are designed in text-format and can be controlled at run-time. Thirdly and lastly, such systems, even based on OO components, can enjoy features from the agent side.

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” [3]. Sending and receiving messages are the two main activities of agent. JADE [4] (Java Agent DEvelopment) framework is one of the platforms that conforms to FIPA [5] standards for developing agents.

2. ARC (Agent/Rule/Class) design framework

2.1 Rule specification

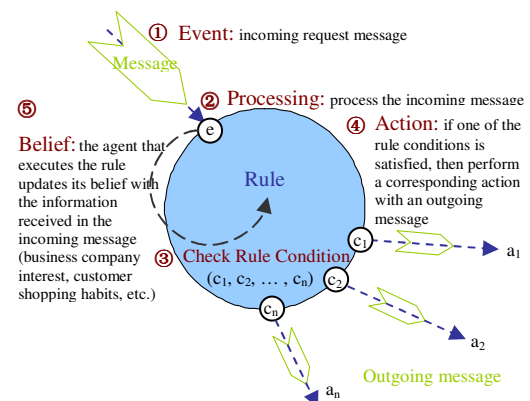


Figure 1, Business rule specification

An agent processes a rule using the following steps, as shown in Figure 1.

1. Check event – find out if the rule is applicable to deal with the perceived event.
2. Do processing – decode the incoming message, construct business objects to be used in later phases.
3. Check condition – find out if the (condition c_i) is satisfied.
4. Take an action – if c_i is satisfied, then do the corresponding (action a_i) that is related with (condition i) as defined by the rule. Then send a result message to another agent (possibly the triggering one). If c_i is not satisfied, then go back to Step 3 and check the condition c_{i+1} .
5. Update beliefs – according to the information obtained from the message just received, the knowledge of the agent to the outside world is updated.

Business rules, as we specify here, make agent another abstraction over object. An agent uses a dedicated rule for a specific task and, in turn, a rule uses business classes to complete it. What and how classes are to be invoked can be specified in rules and these are configurable. The mutable requirements on components collaboration can be externalised in rules and this reflects in agent knowledge in terms of their collaboration partners, events processing, and response message. Different actions can be set in rules as reactions to different conditions, in an order of user preferences/priorities.

We provide two models that enhance the traditional UML models and integrate business rules into these for the purpose of designing agent systems based on the ARC design framework.

2.2 Structural Model

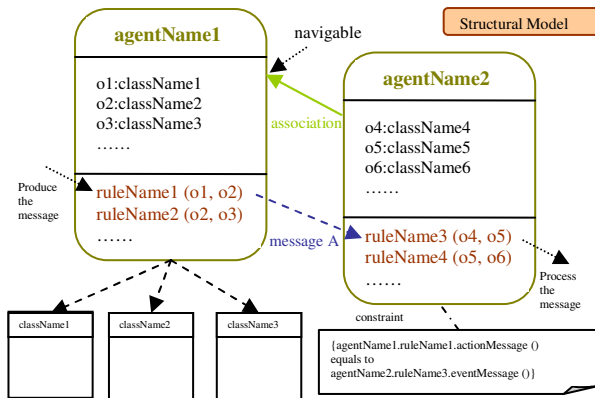


Figure 2, ARC structural model

An ARC structural model diagram has the Class Diagram, the backbone of UML as its counterpart in the OO models. Agents are regarded as superior to classes in this model, just like classes are regarded as superior to attributes in OO models. 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 behaviour of the agent.

In such a model, agents are connected by the “association” line, directed from the message receiver to the message sender. Standard methods invocation between classes is replaced by rule collaboration between agents, where one rule produces a message and the other processes it. What and how components are to be invoked can be configured with rules and such configuration information is obtained by agents at run-time to ensure the deployment of most up-to-date requirements. Eventually the requirements on components interaction are replaced by agent interaction and such knowledge is formatted in business rules. Another layer of abstraction is hence achieved.

2.3 Behavioural Model

ARC structural model diagrams draw pictures of the system in static relationship between its composite elements. Like UML Class Diagrams are complemented by Sequence Diagrams for behavioural modeling, ARC behavioural model is designed to capture the behavioural scenarios. Inspired by the Agent-Object-Relationship model [6], we group associated agents/rules/classes and show how their behaviour can finally achieve certain business goals. A rule is considered as an event-driven processing unit for agents in such a model. A behavioural model diagram corresponding to Figure 2, assuming that the response message is sent back to the initial agent, is shown in Figure 3. It visualises the actual system function in a sequence of actions, with only the satisfied (condition, action) couplet shown and previously unsatisfied couplets ignored.

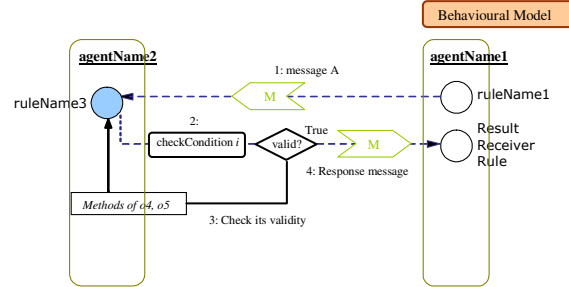


Figure 3, ARC behavioural model

In traditional OO systems, objects are aware of which other objects they will pass messages to, but are unaware of which objects will pass messages to them. Unlike these, ARC framework achieves full architecture independence (two-way encapsulation), which requires that the detail of where objects will send messages should also be hidden [7]. Such an achievement results from managing agent collaboration with rules, which are not hard-coded but configurable, in ARC behavioural model diagrams. Thus, agent collaboration partner and the way of collaboration can be amended, according to changing requirements.

3. ARC design framework benefits

As it has been mentioned in the sections of structural model and behavioural model, the ARC framework introduces another layer of abstraction over traditional OO components and achieves full architecture independence. The consequence of these is that business rules, sitting between agents and business classes, are used to manage system collaboration, and become the knowledge source of agent behaviour.

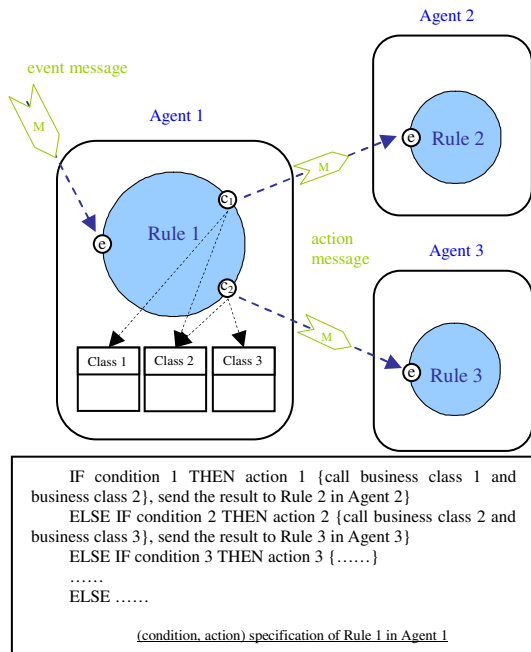


Figure 4, Using ARC framework for adaptivity

A major outcome of applying the ARC framework to build agent systems upon OO systems is that adaptivity is achieved in the resulting system. As shown in Figure 4, a collection of (condition, action) pairs can be set for Rule 1, specifying on receipt of an external event, usually a request of service, the actions to respond in different conditions. Re-matching of condition and action can change the action for the same condition to another pre-set action. To give an example of this, suppose condition 1 is satisfied initially. The exchange of action 1 and action 2 for condition 1 and condition 2 can make Rule 1 invoke Class 2/3 instead of Class 1/2, and send the result to Agent 3 instead of Agent 2. Calling alternative business classes in actions will change the system behaviour. Suppose method 1 in Class 1 will be invoked by action 1, and now another version of Class 1 is available with method 1 updated, and then re-configuring action 1 to relate with the new version of Class 1 can introduce new behaviour without touching any code. Re-ordering of the conditions can change the priorities for applying the corresponding actions, if conditions are not all exclusive. If both condition 1 and condition 2 are satisfied, then the exchange of them in the Rule 1 specification can make Agent 1 execute action 2 instead of action 1. Therefore, agents are both adaptive

internally because they can make use of different business classes and also externally because they can speak to different collaborators differently due to the fact that information is retrieved from adaptive rules. Moreover, all these settings are usually text-based and can be configured by business people through simple user interface.

One of the other uses of ARC framework is that it can play the role of middleware. Suppose Class 1/2/3 used by Agent 1 are developed in Java and a set of classes used by Agent 2/3 are developed in C. The cross platform agent layer enables the inter-operation between component systems written in different programming languages. (This is a common requirement of distributed software architectures like CORBA).

Additionally, some agents can be dedicated for modularising crosscutting concerns. For example, an agent with a set of authorisation rules, an agent with a set of security rules and an agent with a set of logging rules can be developed. They are ready to listen to the event messages requesting such services and reply with the results, such as granting authorised access, read/write operation permission or logging of events. The separated central modules of these can minimise code tangling and code scattering. The use by demand feature of these by ordinary agents make the evolution of them the evolution of crosscutting concerns in the whole system. (This meets an aim of aspect-oriented programming.)

4. Implementation & Deployment

4.1 Case study

To illustrate the application of ARC framework, we introduce an e-commerce case study. Suppose that a retailer supplies goods to customers from various supplier companies, who may or may not service the retailer. Overall, the relationship between the retailer, customers, and supplier companies can change at any time. Basic business classes have been developed and new versions of them may be introduced in the unknown future. The replacement of them is required to have no effect on the running system. Companies also want their decision making process be flexible, while they process customer orders delivered through the retailer. They may accept the order, reject it, or forward it to other companies, depending on different conditions.

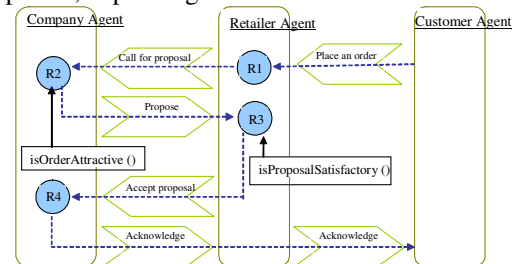


Figure 5, ARC behavioural model for the case study

Of the two design models of ARC, structural models are the basis. Behavioural models are built upon them, and it is the behavioural models that we use for the later implementation. Such a model for the case study is shown below in Figure 5. In this example, an order is placed by a customer, processed by the retailer who, in turn makes a call for proposal to a company. The company replies with a proposal, if the order is attractive and the retailer accepts it, if the proposal is satisfactory. The process completes with an acknowledgement from the company to the retailer who then acknowledges the customer.

4.2 Business rule implementation

As it is shown in the rule specification in Figure 1, one rule is composed by five elements. These are encoded as respective tags in XML. They include details about the triggering event, the processing of the event, a series of (condition, action) couplets and the priority. The XML representation (The XML Schema can be inferred) for rule R2 is shown in Figure 6.

```
- <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>
```

Figure 6, The XML definition for rule “saleProcessing” (R2 in Figure 5) owned by “CompanyAgent”

XML-based rules have the precise definitions of agent behaviour, complementing ARC models. This is what the UML Diagrams lack [2]. In general, each agent reacts to the receipt of a message by executing a rule in the following way.

1. Get a list of its managed rules from a rules document according to the <owner-agent> section.
2. Filter these rules and retain those which are applicable to the current business process according to the <business-process> section.
3. Get the rule currently has the highest priority according to the <priority> section.
4. Check the applicability of this selected rule, that is, if the <event> section matches the event that has occurred. In other words, check if the agent that triggers the received message is the same as that given in the <from> section of the <message> in <event>, and the received message format is also as specified in the <message>. If that is not the case, go to Step 9.
5. Decode the message received and build business objects from it following the <processing> instructions. Constructor methods of existing classes will be involved. Global variables declared in the <global-variable> section will be used to save the results.
6. Check if the current condition specified in the rule is satisfied according to the <condition> section. Constructed business objects will be involved, and their methods will be invoked upon to assist the rule to function. If the condition is not satisfied and it is not the last condition, check the next condition, otherwise go to Step 9.
7. Execute the corresponding <action> section. This involves encoding constructed business objects that refer to <global-variable> into a message. Send the message to the agent which is specified in the <to> section of the <message> in <action>.
8. Analyse the business object which has been decoded from the message received and update the agent’s beliefs with the new information available.
9. Remove this selected rule from the rules set obtained in Step 2 and go to Step 3.
10. Wait for the next event.

The required adaptivity in the case study can be satisfied in two aspects.

1. The developed business class “Order” can be replaced by “newOrder” with the new isOrderAttractive() method achieving a different effect, reflecting new business policy used for order evaluation processing. The configuration of the rule for replacement

of business classes can update the agent behaviour without changing any code in the system.

2. Different collaborators of the company agent can be specified when it receives an order. Three (condition, action) couplets can be defined. The default condition is that the order is attractive, then it proposes price of the order, dispatch time and so on to the retailer agent. For simplicity, this is the only couplet shown in Figure 5 and Figure 6. In other situations, the company agent may reject the order proposal if the customer/retailer has bad credit. Alternatively, forward it to other company agents, if it finds the required goods can be provided to other retailers at a better price. The configuration of these is very simple via the XML-based rule. For example, changing the rule field of <action>/<message>/<to> can change the collaborator of the agent that owns the rule. Additional (condition, action) couplets can be appended to accommodate upcoming (condition, action) pairs.

These two aspects represent intra-agent and inter-agent adaptivity, respectively, both achieved by the use of business rules.

4.3 Tool support

A tool has been developed to support the specification of ARC behavioural models and business rules. Figure 7 captures a window from this tool showing the construction of model diagrams in its main panel and the definition of rules via a user-friendly tree structure.

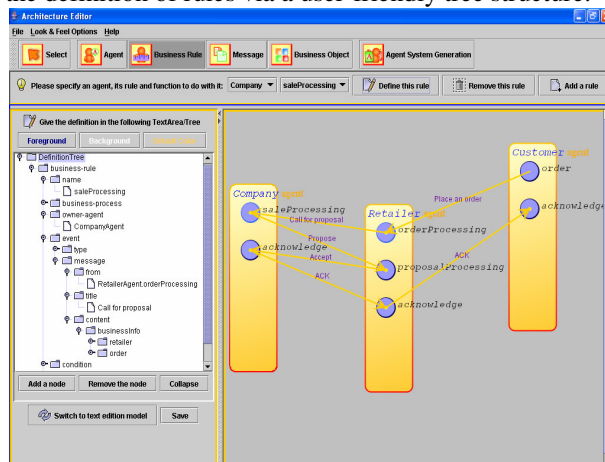


Figure 7, Supporting tool for ARC framework

Each rule specified in the tool maps to one agent behaviour. XML-based rules are translated by agents at run-time. Pseudo code of the agent behaviour represented by R2 in the case study is shown in Figure 8. A shared module called “Rule” (It is part of the JavaBeans in Figure 9 which we will talk about later) facilitates agents to translate XML-based rules as their behaviour. The module accesses the XML definition of rules and can be used to assemble corresponding objects. The methods `getPriority()`, `getEvent()`, and `getAction()` are provided by “Rule”.

```
thisAgent.addBehaviour (Rule thisRule) {
    thisBehaviour.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.currentTimeMillis (),
                    fromAgent, m);
            }
        }
        m = thisAgent.receiveMessage ();
    }
}
```

Figure 8, Pseudo code of agent behaviour

4.4 Deployment

As soon as ARC models are specified graphically in the supporting tool with the business rules specification in XML, agent interaction models, rule reaction patterns and message flows are established accordingly. The actual agent system that will run on the JADE platform in a distributed network is initially generated from the tool. A central XML-based business rule repository is deployed in the network, containing the rule definitions and the registered business classes that are used by rules. A JavaBeans component is implemented, responsible for parsing the XML format of business rules and presenting the parsed business knowledge in the tool. The tool is continuously used by business people to maintain business knowledge. The edition through the tool for the knowledge change is saved in the XML repository using the same JavaBeans.

All agents access the repository via the JavaBeans as well, in order to obtain the most up-to-date knowledge in an easy to operate format. In the beginning, each agent has the knowledge of whom and how they will collaborate with, dictated by the initial rules. While the system is running, the Business Knowledge Model can be continuously under maintenance through the tool. Then the generated Agent Model can be updated with new requirements knowledge. Eventually agents can always get the desired behaviour as soon as it has been

specified through the tool, and can be continuously updated. The deployment of such an implemented system is shown in Figure 9.

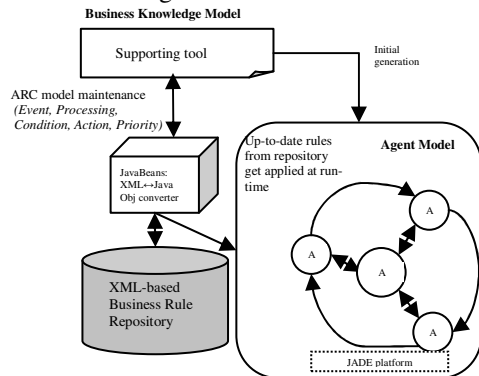


Figure 9, Deployment of agent systems based on ARC design framework.

5. Related work

The ARC design framework for the agent system is built upon existing OO components. The main outcome of modelling agent knowledge in business rules by using ARC is adaptivity.

Recent approaches aimed at software adaptivity are the strategy design pattern [8], the Coordination Contract [9], and the Adaptive Object Model (AOM) [10] [11]. One fundamental property of all these approaches is that they are OO-based. Since objects are passive and traditionally have fixed methods, to make them adaptive, either it is inconvenient or impossible. For example, in the case of the strategy design pattern future behaviour must be fully predictable. For the Coordination Contract approach only inter-component collaboration can be adapted and so it is insufficient. The AOM leads to an architecture which is hard to understand and maintain.

Agents have, for example, been proved to be useful for bringing dynamics, flexibility and adaptivity to travel planning [12], coordinated product development and manufacture [13] and manufacturing systems control [14]. There has been previous work [6] [15] on modelling business rules (or policies) in agent systems. However, these are not easy to implement and their rules are not configurable at runtime by business people.

The contribution of the ARC framework is in using business rules between agents and classes, so that, original classes are kept intact most of the time as stable components. Therefore, an easy way is provided to configure rules, which make use of classes differently according to the configuration. Agents are empowered to behave according to rules specification at run-time and, through these, flexible system behaviour is achieved.

6. Conclusion & future work

Work on defining the ARC framework is in three areas. i) Building agent systems on top of OO systems

using the ARC framework achieves adaptivity by configuring host system components collaboration. ii) Inter-operability is achieved by enabling the collaboration between cross platform components that are written in different languages, in a distributed environment. iii) Easy maintenance is facilitated by separating crosscutting concerns in dedicated agents.

Overall, the potential for adaptivity is promising especially since the benefits of the OO paradigm are not abandoned and the features of CORBA and aspect-oriented programming have been introduced via an extra layer being agents coupled with knowledge in business rules. The ARC framework will be made more powerful, but work so far indicates that it will contribute in a novel and substantive way to the business need for adaptivity in systems.

7. References

- [1] Kleppe, A., Warmer, J. & Bast, W., "MDA Explained: The Model Driven Architecture: Practice and Promise", Addison-Wesley, 2003.
- [2] Fowler, M., "UML Distilled: A Brief Guide to the Standard Object Modeling Language" 3rd Ed., Addison-Wesley, 2004.
- [3] Jennings, N.R., "On Agent-Based Software Engineering", Artificial Intelligence, 14(2) 145-189, 2000.
- [4] JADE platform, <http://sharon.cse.it/projects/jade/>.
- [5] Foundation for Intelligent Physical Agents, <http://www.fipa.org/>.
- [6] Wagner, G., "The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior", Information Systems 28(5), 2003.
- [7] Object Management Group, Inc., "Applying UML 2 to Model-Driven Architecture", 250 First Ave. Suite 100, Needham, MA 02494, USA, 2003.
- [8] Gamma, E., Richard, H., Johnson, R. & Vlissides, J., "Design Patterns", Addison-Wesley, 1994.
- [9] Andrade, L. & Fiadeiro, J.L., "An Architectural Approach to Auto-Adaptive Systems", 22nd International Conference on Distributed Computing Systems Workshops, 2002.
- [10] Yoder, J.W., Balaguer, F. & Johnson, R., "Architecture and Design of Adaptive Object-Models", ACM Sigplan Notices, 36(12) 50-60, 2001.
- [11] Yoder, J.W., Johnson, R., "The Adaptive Object-Model Architectural Style", Proc. 3rd IEEE/IFIP Conference on Software Architecture: System Design, Development and Maintenance, 3-27, 2002.
- [12] 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(20) 211-222, 2004.
- [13] Jia, Z.H., Ong, K.S., Fuh, J.Y.H., Zhang, F.Y. & Nee, A.Y.C., "An adaptive and upgradeable agent-based system for coordinated product development and manufacture", Robotics and Computer-Integrated Manufacturing, 20(2) 79-90, 2004.
- [14] Goh, T.W. & 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.
- [15] 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.