

A Hierarchical Agent-oriented Knowledge Model for 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*

Abstract

Software systems used in a business environment must consider knowledge about business processes, business rules, business terms, and their manageability, adaptability, and maintainability due to changing environment over time. In this paper, from a Software Engineering perspective, we propose hierarchical structure for business knowledge that is then made available to agents running in systems. Such knowledge is sourced from the business requirements and becomes a knowledge base of agents, the basis for agent behaviour. Because the knowledge is externalised from the system, system behaviour is easily maintained. As a result, the hierarchical agent-oriented knowledge model is advanced and advocated. The work extends the framework of Agent-Rule-Class (ARC) [2] and contributes a further development of the Adaptive Agent Model (AAM) [3] and is illustrated using the case of a rail track management system.

1. Introduction and Background

Software must change if it is to remain useful [1]. When knowledge of business requirements is structured in a manageable and maintainable manner, the effort in re-delivery by developers is reduced. Better still, when software agents are used to manage and apply the configurable knowledge, software adaptivity is facilitated [2] [3]. The authors of the paper view agents from a Software Engineering perspective, where agents are computational entities that have dynamic behaviour, being situated in a changing environment.

In practice, common agent-oriented software development approaches, as reviewed in [4], extend two basic approaches: object-oriented (OO) approaches, where agents are considered as active objects, and knowledge engineering (KE) approaches, where agent knowledge is modelled. Most current approaches only focus on one of them. An example approach of the former type includes agent-oriented programming (AOP) [5], an analogy of object-oriented programming. Another typical example is an agent-oriented methodology for enterprise modelling [6], which combines OO methodologies and enterprise modelling methodologies.

In the case of the latter approach, the knowledge acquisition process is the focus as in the case of agent knowledge modelling, CommonKADS [7] being a European standard. Other related approaches include Agent Oriented Abstraction [8], agents being used in corporate knowledge modelling, and DIAMS [9], where agents are used to address knowledge management and information access issues within distributed business environments.

Inheritance and extension in either direction is useful and promote methodology reuse, but insufficient in our opinion. In answering an open question of common interest, “why are agent-oriented methodologies necessary”, we argue, using the proposed Adaptive Agent Model (AAM), such a methodology benefits both communities. From one perspective, it provides a higher level of abstraction than an OO methodology on its own. Thus, knowledge can be externalised for easier management rather than fixed in objects. From another perspective, the deployment of modelled knowledge is supported by an underpinning object layer.

In AAM we aim at an operational development paradigm, by which developers of traditional OO systems can easily transfer and adapt their skills, while the resultant agent-oriented systems are easier to manage and deploy domain knowledge. For this purpose, knowledge is encapsulated in business models, being captured requirements structured to be executable by agents. The running agent systems make use of a lower layer of object components, the use of which is specified in the knowledge model in an adaptable form. By using the knowledge in a structured hierarchy, agents easily know what, when, and how components are to be used in various levels. The ultimate goal is to produce reusable and executable models external to agents, AAM being a concrete instantiation of Model Driven Architecture [10].

2. Hierarchy Overview

This section briefly investigates typical business systems and how their requirements, if structured as knowledge for agent systems, give rise to distinct layers emerge. Figure 1 illustrates the three layers of business knowledge.

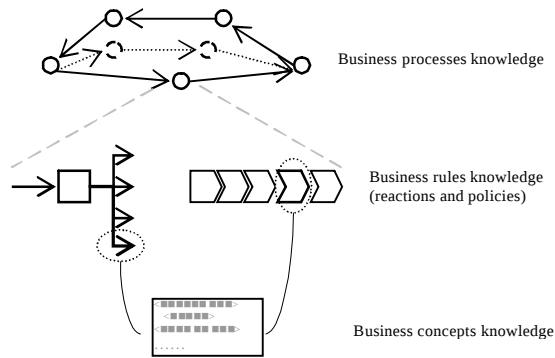


Figure 1: AAM knowledge hierarchy

As shown, in the *business processes knowledge* layer, agents act and react in collaboration one another, relying on their belief or interaction protocols. This first layer is built upon the architectural collaboration patterns of agents. In this layer, inter-agent message exchanges and the results of these satisfy the goals found in the business requirements. A business process involving human or computer agents describes the knowledge in this layer. The next layer of the knowledge model is the *business rules knowledge*. This is found in individual agents and captures how each agent individually performs business tasks in a business process. This involves internal decision making process and execution of proper business policies in that context. By implication, this knowledge must be modelled and known to the agents. Figure 1 shows reaction strategies and policies at this level. These are both categorised as business rules, according to which agents must comply to behave. The third layer of the knowledge model is *business concepts knowledge*. Here, the vocabularies agents use to communicate with each other in the systems, which also might be used in the policy specifications, are defined as building blocks at the bottom of the hierarchy. These building blocks are called business concepts in AAM. For explanatory purposes, AAM components and their features are further described in Table 1.

Table 1: AAM components

Component	Description
Agent	carries out activities required by AAM knowledge database
AAM knowledge database	- machine understandable, and thus can be translated by agents - built from business requirements
Business process	- principle component of knowledge database - involves other components, can be decomposed into sub-processes as specified in rules
Business rule	- statements, actions, and procedures that should be enforced in business environment - business requirements in a form suitable to be assigned to individual agents - refers to business concepts
Business concept	- essential and irreplaceable atomic business units truly exist in business environment - referred by business rules and spoken by agents

3. Case Study

An actual national railway system specification has been investigated as our case study. The system is mainly responsible for the running of a railway on a daily basis, monitoring train running with regard to incidents and ensuring the safety of the train services by conveying issues to relevant parties for resolution. Being a very complex system, the specification document has more than 250 pages and contains a large number of functions with detailed descriptions. A selected excerpt of the specification follows in Figure 2, with a concern about fault management of the railway system.

Case background: The specification has a main area of **Train Running**, and another **Infrastructure Management**, both are subdivided into *Business*, *Incident*, and *Execution* domains. Domains relevant with fault management include: Infrastructure Management - Incident (abbreviated **IMI**), being responsible for passing of information about faults between the system and contractors; Infrastructure Management - Execution (abbreviated **IME**), being responsible for granting of isolations; Train Running - Incident (abbreviated **TRI**), being responsible for refinement and corrections of planned train journeys. External entities: **Train Operators**, who initiate train running requests, and have to be consulted when dealing with perturbations, and **Contractors**, who carry out maintenance.

Case terminology: The infrastructure of the railway system consists of the assets necessary to run the trains. An *infrastructure asset* is any identifiable item of interest within the infrastructure. An infrastructure asset may have a number of asset *faults*. Asset faults may either cause an *incident* or may be caused by an incident. An incident may cause a *track restriction*. Under a *contract* or a variation to a contract with a contractor, infrastructure assets are maintained and asset faults are fixed.

Case description: An asset fault is either reported to the system (Requirement: *IMI-AcceptFaultReport*) or detected directly by the system (Requirement: *IMI-NoticeFault*). The handling of both cases is the same (Requirement: *IMI-HandleFault*). If the fault has already been cleared no further action is needed immediately. Otherwise the system notifies the Contractor responsible for the fault and agrees a priority for fixing the fault. The fault may not require immediate attention and may have no immediate impact, in which case nothing further is done. If the fault does have some impact an incident is recorded. It may be necessary to put in place immediate track restrictions (Requirement: *IME-ImposeSuddenRestrictions*), and this will involve changes to forecast train journeys (Requirement: *TRI-RespondToIncident*). Affected train journeys are amended for re-scheduled services to the Train Operator.

Figure 2: Excerpt from rail track case study

4. Layered Knowledge Modelling

The three layers of the knowledge model described in Section 2 are given more details in this section.

4.1 Conceptual Model & Fact Model

The use of metadata and ontologies has become a predominant element in Semantic Web research. These concepts prove useful for management of meta-information separately from the applications that gave rise to much of that information in the first place [11].

For the purpose of easy agent knowledge maintenance, and to ease interoperability, we use a *Conceptual Model* (CM), which externalises business concepts from the applications that use them. This provides a way for agents to understand available terms and their relationships within documents or messages. Concepts are also used to construct meaningful business rules, applicable in various situations.

It is a common practice in the object-oriented community to begin the modelling process with a conceptual model of the domain space relevant for the application being designed. One may use a grammatical analysis of natural language description of a system to discover objects to build the system. This technique is also plausible for the identification of business concepts and the building of a taxonomy. However, here they are not turned into system components, rather, they become knowledge of agents and vocabularies of conversations.

Example business concepts found in the case study description are “fault”, “incident”, and “restriction”. A “fault” has properties indicating its location, impact, and priority, these themselves being business concepts. All business concepts and their relationships in an object-oriented structure form the ontology of our business models. These must be registered in the CM before being referred to by business rules and business processes. Relationships between business concepts may be enforced as required by business rules for business needs. An implication of a relationship between “fault” and “incident” is such a case. Business concepts are represented in XML. The example of “fault” and its related properties is shown in Figure 3.

```
- <concept>
  <name>fault</name>
  - <properties>
    <property>type</property>
    <property>location</property>
    <property>impact</property>
    <property>priority</property>
    <!-- ... more properties ... -->
  </properties>
</concept>
```

Figure 3: XML representation of a business concept

At runtime, concrete facts are established in a *Fact Model* (FA) as instantiations of abstract concepts. For example, when a report about an asset fault arrives, one fact may be established and states that a fault has occurred in London, and is a class of “rail broken”, and so on. Properties of this business object are thereafter populated with values. One dedicated agent, the *Fact Manager Agent* (FMA) is responsible for the management of all facts at that moment. It interacts with a *Policy Rule Manager Agent* (PRMA) to deduce new facts from existing facts by application of *Policy Rules* (PR), and make available to all agents the known facts. Once facts are established as a result of information brought by event messages, agents have knowledge to reason about their reactions using *Reaction Rules* (RR).

Agent knowledge gets dynamically updated as message exchange continues and facts are added or removed. The PRMA inherits the original BRMA of its mechanism [2] [3]. Section 4.5 describes how these special management agents and ordinary agents work together.

This methodology is supplemented by a lower layer class facility which enables the use of an existing OO infrastructure. The facility is based on an agent-class hierarchy [2], where dynamic agents invoke static class methods as determined by business needs. At runtime established facts are mapped to business objects instantiated from business classes, the schemas of which are as defined in the CM. Methods defined on the business objects are invoked for the manipulation of facts as business rules permit or require. This leads to the availability of additional knowledge, supporting agents in their reasoning and behaviour. Because the business concepts that comprise the business rules are separated from the classes which are associated with them, it is only at the time when they are used that the specific matched class methods are bound. Therefore, classes to be invoked at runtime are exchangeable and new behaviour can be achieved by the replacement of classes or their methods.

4.2 Policy Rule Model

Business policies change over time and so externalisation of them as executable rules is justified. Business policies structured in an IF-THEN format is presented in [3] as global rules that all agents in the system should obey. A Policy Rule captures a constraint or invariant. PR assertions on the logical relationships between entities must always be true to reflect the required policies. The compositional entities of PR are business objects, attributes, associations, operations, and the compositional operators of a PR are: IF, THEN, AND, OR, and so on. The IF condition of a PR returns a boolean expression over relationships between entities or values. The THEN action of a PR is an assignment of entity values, concrete values, or acts to other entities or actors. Typical PRs arising from the scenario of fault management are categorised as follows and illustrated with examples.

i). Policies defined for classification of business objects, based on facts.

Rule1.
If fault is located at the capital cities
Then it has “*immediate impact*”

This knowledge is stored as shown in Figure 4.

```
- <policy>
  <id>100</id>
  <condition>
    fault.location == "London" OR "Edinburgh" OR
    "Cardiff" OR "Belfast"
  </condition>
  <action>
    fault.impact = true
  </action>
</priority>5</priority>
```

</policy>

Figure 4: XML representation of a PR

ii) Policies defined on the relationships of (classified) business objects and their attributes, facts deduced.

Rule2.
If fault has “immediate impact”
Then it has a high priority of 10

iii) Policies defined on the relationships of (classified) business objects/attributes and behaviour of system, behaviour triggered. These contribute to the formation of RRs. Rule3&4 are given here and are part of the requirement *IMI-HandleFault*.

Rule3.
If fault has no “immediate impact”
Then IMI-HandleFault does nothing

Rule4.
If fault has “immediate impact”
Then IMI-HandleFault establishes a new incident associated with the fault AND request IME to place track restrictions

The PR form of IF-THEN essentially states if the IF antecedent clause is true then the THEN consequent clause becomes true as well. New facts are increasingly known and this may satisfy other PRs, which leads to additional facts. Recursively, new knowledge is derived from existing knowledge as PR chains are formed. The process proceeds until no more PR has its antecedent satisfied. A PRMA is responsible for the application of PR wherever appropriate as described previously.

4.3 Reaction Rule Model

In AAM, we define an event-driven agent architecture, agents being responsive to events according to rules. The PRMA is responsible for the application of all Policy Rules when triggered by request events, and thus deduced facts are known and used by ordinary agents. Reaction Rules in contrast represent reactive processes that agents individually follow in response to events. Event-oriented decomposition, based on events that the system must handle, is a design strategy that enables modular architecture, easy design, independent unit development, and eventually effective maintenance [12]. In many agent-oriented systems, agents monitor the occurrence of interesting events and respond to them [13]. The agent response might generate new events to other agents. Many systems view event information as a string without any semantic meaning. In contrast, we consider events as providing the context for agents to react and through sequences of successive events, business processes are executed. Agreements are bound between agents for their interactions using. RRs.

The first step in the agent-oriented development process is agent identification and requirements assignment to agents. Note in case study, IMI, IME, and TRI are labels for business domains with some required functions organised per domain. Suppose one business domain is delegated to one agent, who has the knowledge concerned with that domain. When the

domain is required for different purposes, the corresponding agent responds and plays several roles to realise several aspects of domain functions, in doing so it fulfilling its responsibilities. Interactions among domains are delegated to message passing among respective agents. Such cross-domain interactions require collaboration of agents, and the collaboration pattern of agents is decided by the interactivity of functions of the involved domains. Some agent-oriented methodologies let designers choose to aggregate related roles into a single agent for convenience [14]. In contrast, we believe this should be done at the specification level, where domain division is the most appropriate criteria that decides the nature of agents and their responsibilities.

In the case study, a set of functions are required in the specification related to how the system manages faults. One function of special concern is reconstructed in Figure 5. *IMI-HandleFault*, belongs to the IMI business domain, and constrains IMI in its handling of faults in reactions. This function describes a constrained system behaviour. The used keyword of “is informed by” in Figure 5, followed by the name of another function indicates the source of an event, and “inform” or “use” followed by the name of another function indicates the target of an action. In this sense, a RR specifies the cause and result of an agent behaviour.

IMI-HandleFault is informed by **IMI-AcceptFaultReport** or **IMI-NoticeFault** about an asset fault,
IF the fault has been cleared THEN DO_NOTHING, ELSE
Inform the responsible **Contractor** about the fault with an agreed priority,
IF the fault has no immediate impact THEN DO_NOTHING, ELSE
Create an incident related with the fault AND
Create and put in place track restrictions using **IME-ImposeSuddenRestrictions**

Figure 5: Reconstructed function specification becomes a Reaction Rule

In knowledge modelling of intra-agent reaction processes, we define a RR structure of: {event, processing, {condition, action}_n}. Agents follow RRs for event processing, decision making, and action selection in various conditions. Informative event messages have business objects encoded in them, conforming to pre-defined structures with concrete instantiations at runtime. Specific business objects are decoded and used by the recipient to make corresponding decisions and respond accordingly at the time of running. While different actions are chosen by agents after the decision making, different collaborative agents are chosen, leading to the formation of dynamic business processes, if required.

Essentially, a RR determines on receipt of an event message, after the processing of the message, if certain conditions are evaluated to be satisfied, the actions that agent should perform. Figure 6 shows the XML representation of RR “*IMI-HandleFault*”. Details about steps and the mechanism for executing a RR using such an XML schema have been described in a previous paper

[2]. Supporting tools for viewing, addition, and edition of RRs have been developed in previous work [3].

```

- <reaction>
  <name>HandleFault</name>
  <business-process>Fault Management</business-process>
  <owner-agent>IMI</owner-agent>
  - <global-variable>
    - <var>
      <name>asset</name>
      <type>Asset</type>
    </var>
    - <var>
      <name>fault</name>
      <type>Fault</type>
    </var>
  </global-variable>
  - <event>
    - <message>
      <from>IMI.AcceptFaultReport</from>
      - <content>
        - <report>
          - <reporter>Henry</reporter>
          - <fault>
            <type>rail_broken</type>
            <location>London</location>
            - <asset>
              <id>10015</id>
              <type>rail</type>
              <contractor>Contractor_A</contractor>
            </asset>
          </fault>
        </report>
      </content>
    </message>
  </event>
  <processing>
    asset = new Asset (reportMsg)
    fault = new Fault (reportMsg)
  </processing>
  <condition>
    fault.cleared () == false
  </condition>
  - <action>
    - <message>
      <to>Contractor.FixFault</to>
      - <content>
        - <fault>
          ...
        </fault>
      </content>
    </message>
  </action>
  <condition>
    fault.immeImpact () == true
  </condition>
  - <action>
    - <message>
      <to>IME.ImposeSuddenRestrictions</to>
      - <content>
        - <asset>
          ...
        </asset>
      </content>
    </message>
  </action>
  <priority>5</priority>
</reaction>

```

Figure 6: XML representation of a RR

4.4 Business Process Rule Model

Given the RR structure in the previous section, each RR has an internal processing component, and an external interface of event message receiving and action message sending, through which agents interact. The execution of collections of RRs following message flow sequentially and conditionally forms business processes, and thus is the blueprint of systems. The inter-connected

RRs collectively constrain business processes and form higher level rules for system goals, called Business Process Rules (BPRs). Thus, one RR is about how one task is to be performed following a process, a goal internal to one agent, while one BPR is about how one business is achieved by a compositional process gathered by a whole collection of RRs, a goal shared by many agents. *IMI-HandleFault* is one of the many RRs that comprise a BPR for managing new asset faults, called “Manage New Fault”, shown in Figure 7. Only default conditions are considered as true for simplification.

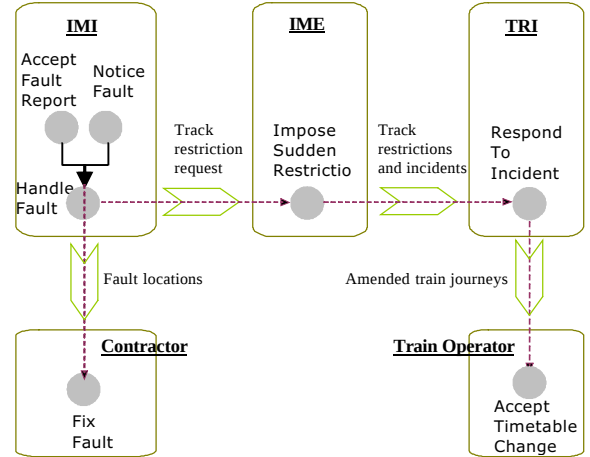


Figure 7: BPR “Manage New Fault” for case study

Agent IMI initialises the above BPR using either of its two RRs: “*IMI-AcceptFaultReport*” or “*IMI-NoticeFault*”, in the interest of solving newly detected faults. The agents finalise the BPR are: *Contractor* and *Train Operator*, the completion of whose functions fulfils the goal of managing new faults. We define Initial Agents (IAs) as those initiate the BPR and Final Agents (FAs) as those finalise BPR. IAs act spontaneously without request by other agents and FAs complete the BPR and request no further action from other agents. Intermediate agents participate in BPRs between the activities of IA and FA during the execution of BPRs. BPRs must be ensured of completeness and consistency to satisfy business requirements of business processes and goals. This means for every input to IAs, results can be expected from FAs, indicating goals of respected business processes are accomplished. As long as the input, output and goals of BPRs are all met, the selection of intermediate agents in participation of BPRs, or their individual decision making and other acts are not concerned. Figure 8 shows the BPR “Manage New Fault” in XML, expecting a new fault as input, and “fault fixed” and “train service re-scheduled” as results.

```

- <process>
  <name>Manage New Fault</name>
  <goal>a new fault is managed</goal>
  - <IAs>
    <IA>IMI</IA>
  </IAs>

```

```

- <FAS>
  <FA>Contractor</FA>
  <FA>Train Operator</FA>
</FAS>
<cause>a new fault is reported</cause>
- <effects>
  <effect>
    A Contractor will fix the fault
  </effect>
  <effect>
    Train Operator will re-schedule train services
  </effect>
</effects>
</process>

```

Figure 8: XML representation of a BPR

4.5 Complementary Nature of Rule Types

RRs are not fully functional without supplying additional knowledge from which to reason and make decisions. Equally PRs cannot work independently without a context. In fact they do complement each other in the overall view of our integrated knowledge models, underpinned by the Conceptual Model, and supporting the realisation of BPR and corresponding goals. Collaborative agents perform recursively in three levels aiming at business goals: BPR-RR-PR. When each agent realises its responsibilities in a BPR, it applies relevant RRs and PRs. The fragment of the BPR in Figure 7 carried out by IMI is conducted in practice as follows.

1. A fault is reported to IMI.
2. The “fault” structure encoded in the incoming message matches with the one defined in CM.
3. A fact about a “fault” is established in FM with its location of “London” as well as other information.
4. A business object “fault” is constructed using the schema defined in CM, as well as an “asset”.
5. The RR “IMI-HandleFault” is selected by IMI in this context as its <event> section is specified to handle reported faults.
6. Facts in FM are looked for in relation with the conditions of the RR, to assist evaluation.
7. FMA interacts with PRMA and a Class Manager Agent to seek additional knowledge either by applying relevant PR or invoking related class methods. The fault is known as having impact as a result of its location, indicated by a PR (R1 in Section 4.2).
8. The business objects of “fault” and “asset” established previously are retrieved and encoded in messages. The messages are prepared to be sent to responsible agents to fix faults and impose restrictions as defined in <action> of the RR.
9. IMI sends the message and FMA demolishes invalid facts.

5. Conclusions

A hierarchical agent-oriented knowledge model is introduced, collectively termed the Adaptive Agent Model (AAM). The AAM is built upon two building blocks: a Conceptual Model (CM) used for vocabulary definition and referred to by agents, and a Fact Model (FM), conforming to the CM constructed at runtime according to agents’ current knowledge. Layered rule knowledge supports agent behaviour.

1. BPR: a business process is initialised by the IAs at the beginning, causing a series of agents to react using various RRs, the process being finished at the FAs.

2. RR: an agent chooses a RR to react to after an event in a particular context, makes a decision, selects collaborators, and requests them to carry on the BPR.
3. PR: while a RR is functioning, a set of PRs may become relevant, so forming PR chains which are applied to assist the RR to make the decision or reflect business policies must be enforced in that context.
4. BC: BCs make up structured terms that are used to construct rules, mapping to objects that are invoked by agents.

AAM contributes a complete framework for building agent-oriented business models not only of pragmatic value for applications that require manageable and maintainable specifications, but also can be tailored and adapted for specific use (ARC framework, business models, rule hierarchy, etc.) due to its layered structure.

6. References

- [1] M. Lehman & L. Belady, “Program Evolution: Processes of Software Change”, Acad. Press, London, 1985.
- [2] L. Xiao & D. Greer, “Modelling Agent Knowledge with Business Rules”, Proceedings of the Seventeenth International Conference on Software Engineering and Knowledge Engineering (SEKE’05), Taipei, Taiwan, Republic of China, 14-16 July, 2005.
- [3] L. Xiao & D. Greer, “The Adaptive Agent Model: Software Adaptivity through Dynamic Agents and XML-based Business Rules”, Proceedings of the Seventeenth International Conference on Software Engineering and Knowledge Engineering (SEKE’05), Taipei, Taiwan, Republic of China, 14-16 July, 2005.
- [4] C. A. Iglesias, M. Garijo, and J. C. Gonzalez, “A survey of agent-oriented methodologies”, In J. P. M’uller, M. P. Singh, and A. S. Rao, editors, Intelligent Agents V — Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98), LNAI, Springer-Verlag, Heidelberg, 1999.
- [5] Y. Shoham, “Agent-oriented programming”, Artificial Intelligence, 60:51–92, 1993.
- [6] E. Kendall, M. Malkoun, and C. Jiang, “A methodology for developing agent based systems for enterprise integration”, In D. Luckose and Zhang C., editors, Proceedings of the First Australian Workshop on DAI, LNAI, Springer-Verlag, Germany, 1996.
- [7] G. Schreiber, B. Wielinga, R. Hoog, H. Akkermans, and W. Velde, “CommonKADS: A Comprehensive Methodology for KBS Development”, IEEE Expert: Intelligent Systems and Their Applications, 9(6) 28-37, 1994.
- [8] P. Maret & J. Calmet, “Modeling corporate knowledge within the agent oriented abstraction”, Proceedings of the 2004 International Conference on Cyberworlds, pp. 224-231, 2004.
- [9] J. Chen, S. Wolfe, and S. D. Wragg, “A distributed multi-agent system for collaborative information management and sharing”, In Proceedings of the 9th ACM International Conference on Information and Knowledge Management (CIKM), pp. 382-388. ACM Press, New York, NY, USA, 2000.
- [10] Object Management Group, Inc., 250 First Ave. Suite 100, Needham, MA 02494, USA.
- [11] J. Pollock & R. Hodgson, “Adaptive Information: Improving Business Through Semantic Interoperability, Grid Computing, and Enterprise Integration”, Wiley-Interscience, 2004.
- [12] A. Wasserman, “Toward a Discipline of Software Engineering”, IEEE Software 13(6) 23-31, 1996.
- [13] N. Jennings, K. Sycara & M. Wooldridge, “A Roadmap of Agent Research and Development”, Autonomous Agents and Multi-Agent Systems, 1, pp. 7-38, 1998.
- [14] M. Wooldridge, N. Jennings, and D. Kinny, “A Methodology for Agent-Oriented Analysis and Design”, Proceedings of the Third International Conference on Autonomous Agents (Agents-99) Seattle, WA. 28, 1999.