

A knowledge hierarchy model for adaptive Multi-Agent Systems

Liang Xiao* and Des Greer

School of Computer Science,
Queen's University Belfast, Belfast, BT7 1NN, UK
E-mail: l.xiao@qub.ac.uk E-mail: des.greer@qub.ac.uk
*Corresponding author

Author: Please
reduce abstract
of no more than
100 words.

Abstract: In a changing world the need for adaptivity in software systems is apparent more than ever, since the cost of software maintenance is a huge burden. Adaptivity is needed since business processes, business rules and business terms constantly evolve. This paper argues for a radical solution making use of the inherent adaptivity of software agents. The Adaptive Agent Model (AAM) is described in terms of a hierarchical structure for business knowledge consisting of a business concepts layer, a business rules layer and a business process layer. Collectively, these form a knowledge base that is then made available to agents running in systems. Such knowledge is sourced from the business requirements and becomes the basis for agent behaviour. Because the knowledge is externalised from the system, system behaviour is easily maintained. Making use of a case study, the paper describes how the knowledge hierarchy is made up and assesses its contribution to the goal of software adaptivity.

Keywords: adaptivity; software agent; agent-oriented software engineering; business rule; business process; object-oriented; OO; software engineering; requirements modelling; knowledge engineering; KE.

Reference to this paper should be made as follows: Xiao, L. and Greer, D. (xxxx), 'A knowledge hierarchy model for adaptive Multi-Agent Systems', *Int. J. Computer Applications in Technology*, Vol. x, No. x, pp.xxx-xxx.

Biographical notes: Liang Xiao is a Research Fellow with the Intelligence, Agents, Multimedia Group, at the University of Southampton. He received his BSc from the Huazhong University of Science and Technology (HUST), his MSc from the University of Edinburgh, and PhD from Queen's University, Belfast. He has worked in the Telecommunications Industry as a Software Designer and Programmer. His experience in solving real domain problems has stimulated his interests in the area of software engineering. Specifically, his research work focuses on software adaptivity, agent-oriented modelling and requirements engineering (RE). The results of his research have been presented and published in journals and international conferences.

Des Greer is a Lecturer in Computer Science at the Queen's University, Belfast. He is a graduate of QUB and received his Master's and Doctorate Degrees from the University of Ulster. Before his career in academia at Queens and previously at Ulster, he worked in industry as an analyst-programmer and his research is inspired by real problems in software engineering. His particular research interests are in iterative and incremental software processes, software evolution planning, software risk management and software adaptivity. He teaches software engineering at the undergraduate and graduate levels and is a member of the ACM and IEEE.

1 Introduction and background

Estimates of between 50% and 90% have been made for the proportion of the overall cost of software development spent on maintenance activities (Yourdon et al., 1995). This corresponds to rapidly changing business environments, where business level adaptivity is critical in business efficiency, capitalisation opportunity, cost reduction and revenue generation (Ambler and Constantine, 2000). Lehman's Law of Continuing Change and Increasing Complexity (Lehman and Belady, 1985), confirms the need for adaptivity, but also highlights its complicated nature in stating that after many adaptations, software systems tend to

drift away from their original architecture and design, becoming more and more difficult to adapt, and progressively less and less useful.

One important source of flux in software projects is that the business systems they support are constantly changing. Business requirements and knowledge are often doomed to be incomplete and imperfect, even before starting to build the system. Therefore, the software engineers and developers have to continuously maintain IT systems to align them with the changing business needs. For example, the inherently volatile system requirements such as organisational policies and procedures as well as the business processes applied are among the key factors to be

considered, when one predicts the future system maintenance efforts (Sommerville, 2007). The maintenance burden would be greatly reduced if the system is designed in such a way that it can accommodate new requirements, such as policies, procedures and business processes.

In this paper we will describe an approach that aims to provide the necessary architecture, methods, processes and tools to achieve this goal. The next section will further describe the background. Following that we will describe the AAM used in this research and the knowledge hierarchy used to support it. Section 4 will describe how this layered knowledge modelling is carried out in practice, making use of a case study. Section 5 will examine how maintenance of the models is carried out, again referring to the case study. Section 6 will evaluate this research study with related work. Section 7 summarises the main contributions of the paper.

2 Background

In arriving at a solution to the problem of changing business policies, procedures and processes, we have made the hypothesis that software agents can provide the necessary adaptivity, given the means to react to external requirements change. The purpose of this research, then, is to find a means to capture and model requirements knowledge and to use this, together with the inherent adaptivity of software agents, thus providing an adaptive paradigm for software engineering.

Many attempts to cast agent-orientation as the next major Software Engineering paradigm have been made (e.g., Jennings, 2000; Wooldridge and Ciancarini, 2001). In addition, agents have been found useful on a conceptual level at the requirements elicitation stage (Yu, 1997). A useful survey of agent and agent-oriented Software Engineering can be found in the literature (Tveit, 2001).

Although a general consensus on the definition of an agent is hard to reach, the following has been found useful by many.

“An agent is an encapsulated computer system that is situated in some environment, and that is capable of flexible and autonomous action in that environment in order to meet its design objectives.” (Wooldridge, 1997)

Agents are generally attributed with *autonomy*, *reactivity*, *pro-activeness* and *social-ability* (Wooldridge, 1997; Wooldridge and Ciancarini, 2001), properties associated with the agent definition given above. Usually they are employed with other agents in a distributed and decentralised environment, called a Multi-Agent System (MAS).

The main agent-oriented software development approaches usually extend from one of two basic approaches:

- Object-Oriented (OO) approaches, where agents are considered as active objects
- Knowledge Engineering (KE) approaches, where agent knowledge is modelled.

One example of an OO approach to agents includes agent-oriented programming (Shoham, 1993) which uses an analogy of OO programming. An example of a KE approach is CommonKADS (Schreiber et al., 1994) which focuses on the knowledge acquisition process. The AAM is not limited to either approach, but instead utilises aspects from 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 (MDA) (OMG, 2006).

3 The Adaptive Agent Model (AAM) approach overview

The AAM, as proposed by Xiao and Greer (2005a), recognises that in designing an adaptive software system, the employed software components must be as dynamic as possible, involving least code change. If a software system is expected to meet new requirements dynamically, a means must be sought to externally capture the new requirements and to make them accessible to the running components. Such knowledge represents business needs and, if it is to truly reflect the current requirements in a timely fashion, the knowledge on business processes, procedures, policies and alike must be easily editable by business experts, without intervention of IT experts. In other words there must be a responsibility shift from technical experts to business people in controlling how software systems behave. Business people are evidently the best experts in current business processes and so systems must be designed so that components of the running software can

capture and interpret this knowledge dynamically and immediately, at runtime. Ultimately and idealistically, such a software system may need no re-delivery at all and therefore be without downtime or lost business opportunity due to unavailability while waiting for the next release.

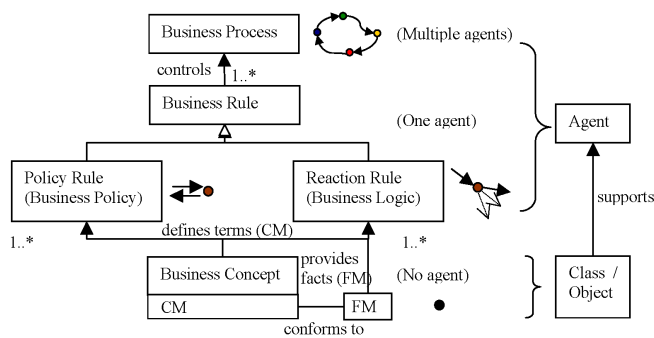
Precisely, it is proposed that two hierarchical structures are combined:

- the hierarchy of business knowledge
- the hierarchy of computing components that use and apply the knowledge.

Collectively, these hierarchies are termed the AAM, as illustrated in Figure 1. The knowledge captured in the business models has two building blocks:

- a Conceptual Model (CM), used for vocabulary definition and rule construction
- a Fact Model (FM), conforming to the CM, constructed at runtime to reflect current fact knowledge.

Figure 1 AAM's two interactive hierarchies: knowledge and component



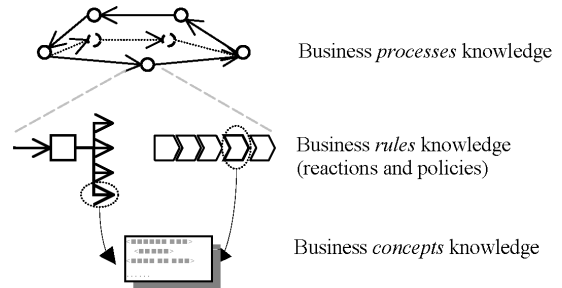
Two rule models are, in turn, used to model agent behaviour. Policy Rules (PR) are global rules that all agents should obey, and describe policies that must be enforced. Reaction Rules (RR) are local rules that specific agents should use and describe reactions that must be performed when triggered by external events. Business Process Rules (BPR) are collections of PRs and RRs that realise business processes aimed at corresponding goals. In addition to the business models, the computing components of agents and objects form another hierarchy, where objects support agents' behaviour. The interaction of the two hierarchies achieves the required goals of the business.

- in the hierarchy on the left hand side of Figure 1, the overall model structure and each individual layer of the model is adaptive to changing business requirements at various levels
- in the hierarchy on the right hand side of Figure 1, the use of appropriate objects by agents is decided dynamically to achieve dynamic running component execution effects

- agent behaviours are driven by dynamically maintainable models, and so the application of one hierarchy by the other is adaptive, maximising their existing combinational adaptivity.

The business models of AAM can be illustrated as follows: Business Processes are controlled by Business Rules, there being two subtypes of business rules, namely PR and RR, both of which are defined in terms of Business Concepts. Together this illustrates how the AAM uses a three layer knowledge hierarchy as shown in Figure 2.

Figure 2 AAM knowledge hierarchy



Essentially, what has been proposed is centred on the hierarchical knowledge model, while agents in the MAS are provided with the knowledge captured in the model at runtime. AAM agents are equally important and no agent is at a higher level than another one. They are separated from the knowledge they make use of and the fact that this knowledge is dynamically editable is the means by which adaptivity is achieved. For example, the reconfiguration of a RR for an agent allows it to change its collaboration partners, internal computation procedure, decision making and message passing. Knowledge on agent behaviour is structured as easily changed business models which can then be executed. Focusing on the knowledge hierarchy, in the *business processes knowledge* layer, agents act and react in collaboration with one another, relying on their belief or Interaction Protocols (IP). 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 an internal decision making process and the execution of proper business policies in that context. By implication, this knowledge must be modelled and known to the agents. Figure 2 shows reaction strategies and policies at this level. These are both categorised as business rules, according to which agents must comply in their behaviour. 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.

4 Layered knowledge modelling in detail

In order to demonstrate the modelling process, an actual national railway system specification has been investigated.

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. The specification document contains a large number of functions each with detailed descriptions, running to more than 250 pages. A selected excerpt of the specification, concerning fault management of the railway system, is given in Figure 3. Using elements from the case study, the three layers of the knowledge model are given in more detail in the following sub sections.

Figure 3 Excerpt from rail track case study

Case background: The specification has a main area of **Train Running**, and another **Infrastructure Management**, both are sub-divided 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.

Author: Please supply right hand side rule for the Figure 3.

4.1 Business concepts knowledge layer

The use of metadata and ontologies has become a predominant element in Semantic Web research. These concepts prove useful for management of meta-information independently of the applications that gave rise to much of that information in the first place (Pollock and Hodgson, 2004). For the purpose of easy agent knowledge maintenance, and to ease interoperability, we use a 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 OO community to begin the modelling process with a CM 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 the 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 OO 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 4.

At runtime, concrete facts are established in a 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 PR, and makes the known facts available to all

agents. Once facts are established as a result of information brought by event messages, agents have knowledge to reason about their reactions using RR. Agent knowledge gets dynamically updated as message exchange continues and facts are added or removed.

Figure 4 XML representation of a business concept

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

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 (Xiao and Greer, 2005b), where dynamic agents invoke static class methods as determined by business needs. At runtime, the established facts are mapped to business objects instantiated from business classes, the schemas of which are as defined in the CM. Methods defined, based 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 Business rules knowledge layer: policy rule model

Business policies change over time and, therefore, externalisation of them as executable rules is justified. Business policies structured in an IF-THEN format, represent global rules, which 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.

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

Rule 1

If fault is located at the capital cities
Then it has 'immediate impact'

This knowledge is stored as shown in Figure 5.

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

Rule 2

If fault has 'immediate impact'
Then it has a high priority of 10

- Policies defined on the relationships of (classified) business objects/attributes and behaviour of system, behaviour triggered. These contribute to the formation of RRs. Rules 3 and 4 are given here and are part of the requirement *IMI-HandleFault*.

Rule 3

If fault has no 'immediate impact'
Then IMI-HandleFault does nothing

Rule 4

If fault has 'immediate impact'
Then IMI-HandleFault establishes a new incident associated with the fault AND requests IME to place track restrictions

Figure 5 XML representation of a PR

```

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

```

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 PRs have their antecedent satisfied. A PRMA is responsible for the application of PRs wherever appropriate, as described previously.

4.3 Business rules knowledge layer: Reaction Rule (RR) model

In AAM, we define event-driven agent architecture as agents being responsive to events according to rules. The PRMA is responsible for the application of all PR when triggered by request events, and thus deduced facts are known and used by ordinary agents. RR, in contrast, represents 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 (Wasserman, 1996). In many agent-oriented systems, agents monitor the occurrence of interesting events and respond to them (Jennings et al., 1998). 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 the case study, that 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 allow designers the choice of aggregating related roles into a single agent for convenience (Wooldridge et al., 1999). In contrast, we believe this should be done at the specification level, where domain division is the most appropriate criterion 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 6. *IMI-HandleFault*, belongs to the IMI business domain, and constrains IMI in its handling of faults in reactions. This function describes constrained system behaviour. The used keyword of ‘*is informed by*’ in Figure 6, 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 agent behaviour.

Figure 6 Reconstructed function specification becomes a Reaction Rule

```

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

```

In knowledge modelling the 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 7 shows the XML representation of RR '*IMI-HandleFault*'. Details about the steps and mechanism for executing a RR using such an XML schema have been published elsewhere (Xiao and Greer, 2005b). Supporting tools for viewing, addition, and edition of RRs have been also been developed (Xiao and Greer, 2006).

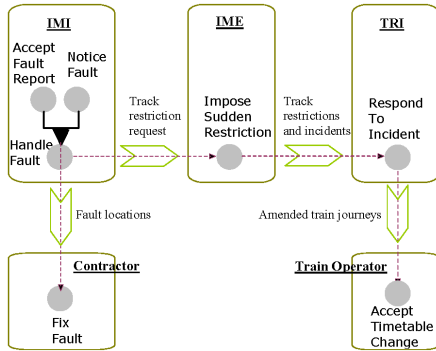
Figure 7 XML representation of a RR

```
- <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>
```

4.4 Business process knowledge layer

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 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 8. Here, to simplify, only default conditions are considered as true.

Figure 8 BPR ‘manage new fault’ for case study



As the figure shows, when one business process is to be formulated in our models, multiple agents are involved and behave collectively in it. Since each agent has its own capability in realising a specific aspect of requirements, the collaboration of agents together contributes to the goal of the single BPR, realising a high level requirement. For example, several agents jointly aim at managing infrastructure faults in the case study. IMI plays an incident management role in the above process. When an incident is noticed, it initialises a handling process that IME starts to

execute. When track restrictions are placed by IME, train journeys must be corrected by TRI to respond to the incident and finally, the train operator receives the change. Individual RRs are executed in the course of events such as the *HandleFault* rule illustrated in Section 4.3 being assigned to the IMI agent for incident management. The building of the business process model, with agents as active components in it, guides the later execution of each part of the process by individual agents. At the requirements level, agents are mapped to business processes to meet the requirements of processes and at the implementation level, agents collaborate together to execute the processes. Therefore, business processes can be decomposed into multiple parts that multiple agents can achieve together in accordance with specific functional specification (as shown in Figure 6). Actually, these decomposed parts are the RR at a lower level of the hierarchy that is assigned to agents individually. This eases the requirements change management, directly maps requirements to agent capabilities, and suitably organises agents around business processes.

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 to 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 initiating the BPR. Final Agents (FAs) as those finalising the BPRs. IAs act spontaneously without request by other agents and FAs having completed the BPR request no further action from other agents. Intermediate agents participate in BPRs between the activities of the IA and the 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 that for every input to IAs, results can be expected from FAs, indicating that the 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 of concern. Figure 9 shows the BPR ‘Manage New Fault’ in XML, expecting a new fault as input, and ‘fault fixed’ and ‘train service re-scheduled’ as results.

Figure 9 XML representation of a BPR

```

- <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>

```

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 CM, 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 RR and PRs. The fragment of the BPR in Figure 9 carried out by IMI is conducted in practice as follows.

- A fault is reported to IMI.
- The ‘fault’ structure encoded in the incoming message matches with the one defined in CM.
- A fact about a ‘fault’ is established in FM with its location of ‘London’ as well as other information.
- A business object ‘fault’ is constructed using the schema defined in CM, as well as an ‘asset’.
- The RR ‘*IMI-HandleFault*’ is selected by IMI in this context as its <event> section is specified to handle reported faults.
- Facts in FM are looked for in relation with the conditions of the RR, to assist evaluation.
- 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 to have an impact as a result of its location, indicated by a PR (R1 in Section 4.2).
- 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.
- IMI sends the message and FMA demolishes invalid facts.

Figure 10 Newly arising policies for the case study

Policy Rule 1: Group Travel IF <i>A minimum of 10 people in a group</i> THEN <i>discount 25% off standard fares, 10% discount on First Class</i> Policy Rule 2: Off-peak Travelcard IF <i>ticket 8:30 p.m. or later from London on weekdays, any time during weekends</i> THEN <i>15% discount</i> Policy Rule 3: Special Offer of today IF <i>ticket for First Class today</i> THEN <i>5% discount</i>
--

Suppose multiple discount offers can be applied jointly, and then when a checkout event occurs, the PRMA selects all appropriate PRs to apply. The customer order information, as well as the original ticket price, is used for the

5 AAM maintenance and adaptation

A business model hierarchy has been established: business concept – policy rule and RR – business process, one layer supporting the next. Also, changes at any layer will spread to layers above it automatically and take effect without the need for human intervention. The change of BCs is instantly live in all business rules that refer to them. As concepts are put into the central repository separately from rules that refer to them, rules weave them together only at the time that agents request the use of them. The other layers co-work in similar patterns. The change to a PR is reflected automatically in a RR, since RRs dynamically choose PRs and form PR chains at the time that agents request the use of them. If the currently selected PR can bring to a RR its changed policy, immediately it is applied. The changes to RRs are reflected automatically in BPRs since they are dynamically composed in an interaction pattern of collaborative agents, specified by RRs collectively.

Referring to the case study, the RR *ImposeSuddenRestrictions*, for example, is involved in two BPRs for managing new faults and for updating existing faults. Once it gets changed, both BPRs start to impose this change in re-scheduling train services from that moment on without any manual change to the BPRs themselves. For each BPR, as long as its input, output and goal are unchanged, the involved intermediate agents are free from participation and can be removed. Agents are activated and rules selected as required by business needs determined at the time of execution.

5.1 Adaptivity of Policy Rules (PR)

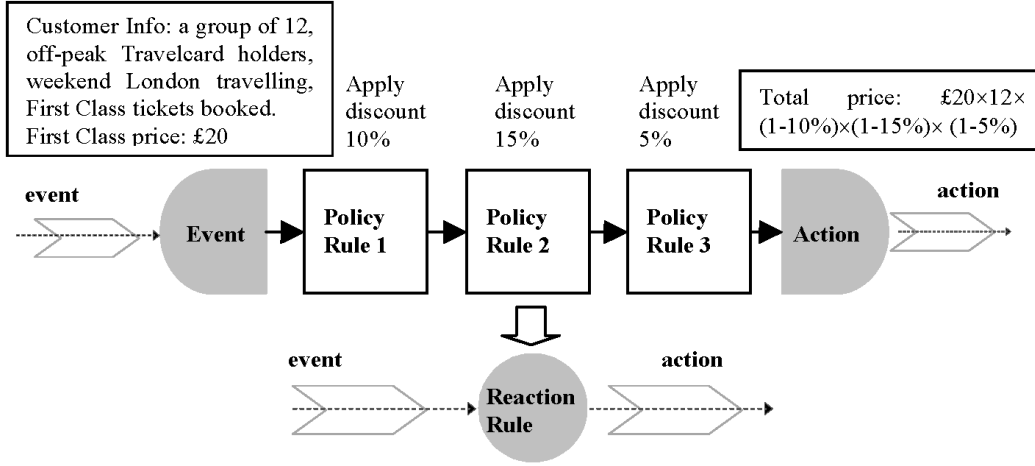
Assume that the rail track system now extends to provide customer services. Included in the extended system there is an online ticket sale system, and the rail track company has the policies described in Figure 10, for selling tickets to customers. For simplification, assume no logical inference relationship among them.

computation of the final price after offer. This process is modelled in Figure 11. Because the pre-condition of each PR shown above is satisfied in this case, they will all be applied and contribute a reduction to the total price.

The final result is the post-condition of the last PR: $\pounds 20 \times 12 \times (1 - 10\%) \times (1 - 15\%) \times (1 - 5\%)$, which will be sent to interested parties, as a consequent action. Five elements, collectively, can be regarded as an integrated

RR with no decision-making branches, where the three PRs compose the <processing> component, while a separate request receiving event and a result despatch action at each end composes the <event> and <action> components.

Figure 11 An event, 3 PRs and an action constituting a Reaction Rule (PR escalates to RR)



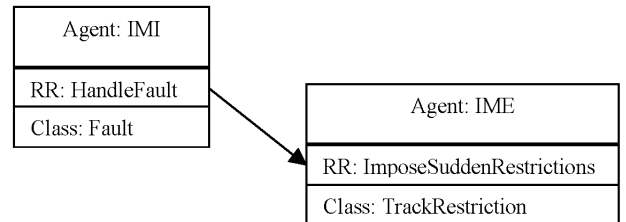
RRs, applicable in local scope to individual agents and PRs, applicable in global scope across agents share the couplet structure of {condition, action}, making their seamless integration possible. If one day it is decided that the three example PRs need to be localised, which means that only one dedicated agent should access them to calculate the discount value, then a RR can be formed to replace the three PRs. Conversely, it is possible to globalise a RR as multiple PRs if such policies captured by it need to be available to many agents. This mechanism provides a means to freely convert rules from being private to being public, assignable to one or many agents.

PRs let the system be adaptive to changing business policies. The addition/removal/modification of one PR has no effect on others, easy edition tools (Xiao and Greer, 2005a) enabling business people to put their current policies into effect immediately. There is no need to re-build the system, because rules are not embedded in code. Instead, agents will flexibly choose from them the appropriate and most up-to-date ones in that runtime context. The goal is to let systems reflect current business policies without re-development. This transfers the application of policies from humans to agents that automatically select and execute, thus eliminating the possibility of omitting to apply any policy. For example, the last PR on a special offer of today will be automatically picked up by the PRMA to execute. On the next day, this will not happen, as it becomes invalid due to the unsatisfied date of application pre-condition. An alternative way of customising the application of PRs is by setting different priorities for them, allowing higher priority policies to override those of lower priority.

5.2 Adaptivity of Reaction Rule

Figure 12 shows the relationship between agents IMI and IME and their collaboration in the Fault Management scenario. IMI uses *HandleFault* to collaborate with IME, which responds using *ImposeSuddenRestrictions*. Two business classes are invoked during the function of two RRs, respectively, in assistance of event processing and decision making. This might cause the illusion that the relationships between agents, the selection of RR, and the selection of business classes are fixed. In fact, the relationship shown is only transient and there is no direct link between agents, or between agents and classes. Rather, such collaborations or association relationships are specified in the selected RR. In other words, it is at the time that a RR is selected for reaction to an event that an agent knows its collaborative agents and supporting classes.

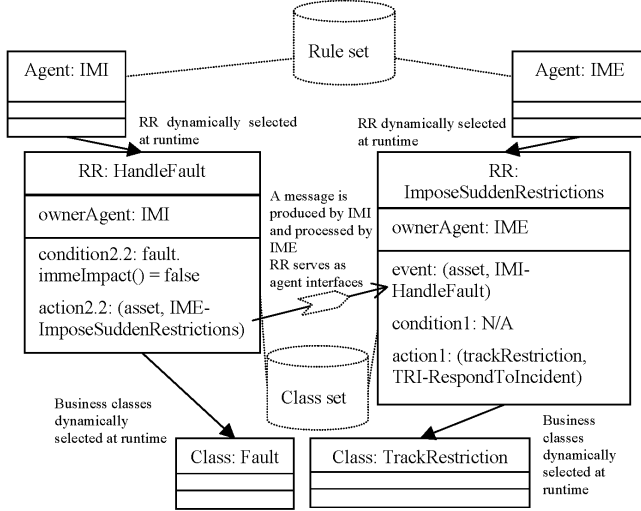
Figure 12 Agent, rule and class in a transient relationship



The configuration of two RRs can change not only the collaboration relationship between IMI and IME, but also the use of 'Fault' and 'TrackRestriction' classes. Such information is completely transparent to agents and only known to them at the time of their activation by events.

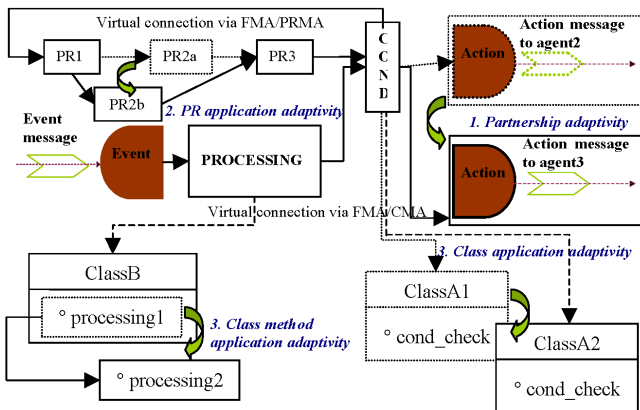
Thus, the agents and classes in relation with each particular agent are replaceable at runtime by configuring RRs, with immediate effect. The accurate relationships among these modelling elements in the chosen part of the case study are shown in Figure 13.

Figure 13 The actual relationships among agent, rule and class



Connections among collaborative agents in business processes are decided at runtime and, thus, the enactment of business processes, as BPR composition depends on decision-making at a set of successive RRs, chosen one after another. A connection is required if and only if an action of one RR points to an event of another RR, because of which a partnership is established between the two agents that own the two RRs. The reconfiguration of {condition, action} pairs leads to a different partnership. This also enables the re-establishment of decision-making tree, as conditions and corresponding actions have been re-built. Figure 14 shows various aspects of adaptivity that RRs can help to achieve, demonstrated through its compositional parts.

Figure 14 A Reaction Rule owned by agent 1 and its adaptivity in three aspects (1–3)



Referring to the labels 1–3 in Figure 14, the adaptivity achieved through edition of reactions rules can be enumerated as follows.

- *Partnership adaptivity.* By changing the ‘action’ part of a RR the partnership for an agent can be changed to being with a different agent.
- *PR application adaptivity.* The PRs that get applied by the PRMA are decided at runtime: a sequence of PRs is established when their pre-conditions are satisfied sequentially, in the same order that PRs are formed. The re-configuration of PR may cause a different sequence of PR to execute at runtime.
- *Class application adaptivity and method application adaptivity.* Which Business Concepts are used is decided at the time when the business rules referring to them are executed. Correspondingly, the business classes and their methods are also determined at that point. The re-configuration of business rules in the use of new business concepts leads to the use of different business classes. Such an adaptive mechanism allows, for example, agents to be able to cope with extendable types of events, since a RR can be configured to use new types of class or new methods in alternative versions of a class.

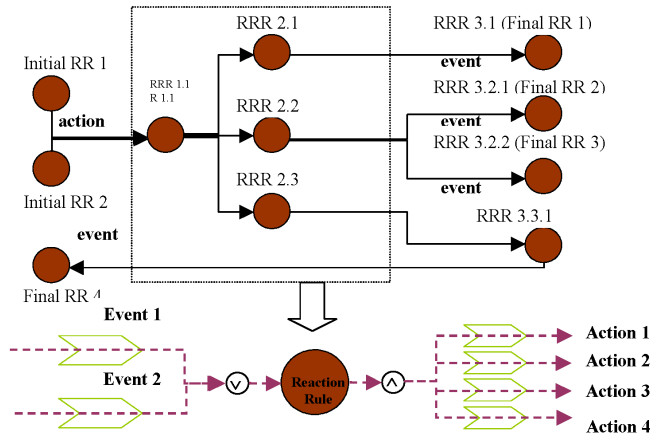
A greater challenge confronted by software engineers today is that software behaviour must adapt in accordance with unforeseen contexts. For example, a new type of event triggers the use of a set of methods in a sequence previously unseen and hence the formation of a new business process built upon new agent interactions which handle the emergent event type. This might mean that a new business service comes into being as required in the real world. Despite the complexity it seems to demand for the handling of such an emergent request, the actual adaptation process is straightforward by using the adaptation mechanisms described above. Briefly, business people can define at the business level a set of new RR interconnected in a BPR. The first RR deals with the initial new event type. Class methods are selected for invocation towards a goal. Appropriate agents are assigned with RR that fit into their domain responsibilities. This grants them the capability to solve additional requirements at runtime as the configuration is carried out without intervening with the running system. Even better would be that an intelligent agent can choose class methods as strategies, design new RR and BPR, look for responsible agents for the additional tasks, and assign rules to agents when it is informed of a new requirement. Such further development would strengthen AAM and grant it greater adaptivity.

5.3 Adaptivity of Business Process Rule (BPR)

A schematic BPR made up of multiple RRs is illustrated in Figure 15. Here, a business goal is realised by the performance of two initial RRs, five intermediate RRs (RR1.1, 2.1, 2.2, 2.3, 3.3.1) and four final RRs. Some computation results could be returned to an initial agent, assuming that RR2 and RR4 belong to the same agent. In that situation, this BPR can be seen as though an

agent uses RR2 to initialise it; with the assistance of other agents, it is finalised with some results, returned to the same agent (and also possibly some others) and received by its RR4. If a BPR constituted by RRs is symbolised as a tree, then the initial RR are its roots and the final RR its leaves. Some of the leaves go back to where the roots grow, but not directly to the roots (initial and final RRs not being the same ones but being owned by the same agents). The tree has many branches going from the roots to leaves; the fact that all leaves can be reached through these branches indicate the accomplishment of the BPR, or the realisation of the business goal of the BPR. Unless all of the leaf nodes in the tree structure are completed, the goal is not realised. Figure 15 also shows the encapsulation of the BPR as a single RR to external entities. As discussed earlier, multiple PRs can be promoted to a single RR as a result of localisation and a single RR can be demoted to multiple PRs as a result of globalisation. The same applies here. Since BPRs and PRs are both applicable in global scope, they may be made private as RRs and assigned to dedicated agents, if necessary, to narrow their application scope. Conversely, since RRs are only applicable in local scope, they may be decomposed into multiple PRs or RRs and so possibly assigned to multiple agents (in the case of BPR) or none, but accessible by any agent (in the case of PR), if necessary, to enlarge their application scope. Overall, the AAM's business model hierarchy of PR-RR-BPR is in unification and one layer can be upgraded or downgraded to another as required.

Figure 15 A BPR tree structure (with OR and AND notation) becomes a RR (i.e., a BPR is encapsulated into a RR)



6 Related work and evaluation

Since traditionally, objects have static structure and fixed behaviour, system adaptivity is hard to achieve using established methods. Even in the agent world, agent platforms currently available to the agent community actually constrain agent system developers in prescribing agent behaviour in agent classes during implementation, an inappropriate circumstance inherited from OO approaches. For example, in the popular agent platform of JADE, IPs described in Agent UML (AUML) (FIPA Agent UML

Website, 2007) are used to represent agent conversations in message sequences for agent system development. IPs are specified using informal notations and are manually translated into program code by agent developers (Ehrler and Craneffeld, 2004). This means that the details of which agents are in partnership, where messages are expected to be received from and sent to, and the processing procedures in response to events, are all frozen once the development is completed. This is not the case using the knowledge hierarchy of the AAM where the structure and collaboration behaviour of agents can be changed after delivery. Other related work attempts to execute agent-oriented UML models include 'Plug-in for Agent UML Linking' (PAUL), which attaches application-specific code to the appropriate points of the protocols for agent execution (Ehrler and Craneffeld, 2004). PAUL recognises that the possible sequences of messages that form agent conversations constrained in IPs need to be interpretable by agents. It uses specific operations that instruct agents the methods to receive messages (as operation parameters) and send messages (as operation outputs), which adds some, though limited, behavioural semantics to the existing IPs. Due to the use of separate code fragments attached to IPs in PAUL, the management and maintenance of them adds an extra burden to the existing system. Since AAM provides editable models to adapt the knowledge in the system there is much less of a burden.

Other Agent-oriented Software Engineering approaches in the direction of using reusable modules such as OO components have been evaluated. Using Agent Patterns (Cossentino et al., 2002) is one way for better code encapsulation and reuse. Agent Patterns reuse patterns of recurring agent tasks to reduce repetitive code. However, a pattern can be hardly reused without change and reuse of patterns in different contexts is not straightforward. State machines have also been suggested for agent behaviour modelling (Arai and Stolzenburg, 2002) and the Extensible Agent Behaviour Specification Language (XABSL) has been specified (Lotzsch et al., 2004) to replace native programming language and to support behaviour module design. Intermediate code can be generated from XABSL documents and an agent engine has been developed to execute this code. Although agent behaviour is modelled in XABSL, it must be compiled before being executed by the agent engine and so is not adaptive. Agent behaviour is modelled as workflow processes in Laleci et al. (2004) and a Behaviour Type Design Tool is described for constructing behaviour. This approach provides a convenient way to compose agent behaviour visually. However, the approach does not offer an agent system generation solution.

All of the above approaches promote module reuse but do not build an architecture suitable for the reuse of appropriate modules. Usually code change is still required, other complexities should be introduced, and the abstraction of agent over object is not fully exploited. In contrast, AAM addresses the problems by associating requirements-captured process-able rules with agents to

abstract agent functions in a comprehensive but manageable manner; allowing the definition of additional rules for agents to make it scalable to accommodate any new requirements; capturing in rules the invocation of generic classes and methods to enable components reuse and easy management. Ultimately, business processes are formed and business goals achieved. At the Requirements Engineering (RE) level, agents are the containers and business models are the knowledge that fills the containers. At the running system level, agents put the captured knowledge into actions. Mapping from requirements to implementation for both the agents and the business models brings traceability to the original requirements. This complete software development process support of the AAM is in contrast to many agent-oriented knowledge modelling approaches. These are usually centred on goal-oriented modelling, including *i** (Yu, 1997), Composite System Design (Feather, 1994), KAOS (Dardenne et al., 1993), Albert II (Heymans and Dubois, 1998) and RE Framework (Donzelli and Bresciani, 2003). Many such modelling approaches give no concern to the integration of their early phase RE models with the rest of software development. This may make the resulting models isolated, not being fully used and eventually losing the advantages originally proposed in their frameworks. Current Process-Awareness Information Systems (PAIS) (Dumas et al., 2005) approaches and tools support either only the design, only the execution, or both but are restrictive in specific types of processes. Such incompleteness hinders an integrated development process for information systems.

7 Conclusions

A hierarchical agent-oriented knowledge model is introduced, collectively termed the AAM. The AAM is built upon two building blocks: a CM used for vocabulary definition and referred to by agents and a FM, conforming to the CM constructed at runtime, according to agents' current knowledge. Layered rule knowledge supports agent behaviour through the following constructs.

- BPRs represent business processes initialised by IAs at the beginning, causing a series of agents to react using various RRs, the process being finished FAs.
- RRs are chosen by an agent in order to respond to an event in a particular context. RRs make decisions, select collaborators and request them to carry on the BPR.
- PRs may become relevant while a RR is functioning. PRs form PR chains which are applied to assist the RR to make the decisions or reflect business policies that must be enforced in that context.
- BCs make up structured terms that are used to construct rules, mapping to objects that are invoked by agents.

The main contribution of the AAM is provided by its knowledge hierarchy and the adaptivity afforded at each layer. Traditionally, in software engineering, all future known conditions must be prescribed before developing the system. However, very often, limited knowledge or control over the environment can be obtained, and not all contingencies can be anticipated. Agents are components that have no perfect knowledge but, rather, an engine that uses an extensive knowledgebase to dynamically perform changing tasks. They allow flexible system architecture. With agents there is the potential that something unknown or uncertain can be fulfilled in the future. In AAM, an agent is an abstraction at the business level and business models are the actual maintenance target that can incorporate new knowledge and contingencies, whenever they become available. Agents make no restriction to structure, behaviour, relationship or decision making. Instead, all these are part of the configurable business models controlled by those who know what they should be and how they should change at different times for different purposes. The overall system has the full freedom to meet the challenge of change in any aspect, even after they have been constructed. All in all, the combination of agents and business models is the natural representation of the business needs and is at the right level where the maintenance should be carried out.

Although the adaptivity of a MAS as a whole does not necessarily require the adaptivity of each of its agents, granting adaptivity to individual agents ensuring that they can all use the most up-to-date knowledge and reflect the current requirements associated with them is the easiest and most straightforward way to achieve this. Nevertheless, both individual agent adaptivity and MAS adaptivity in the AAM have been achieved to some extent. The application of PR, as well as internal decision making and processing in the application of RR, achieves individual agent adaptivity. Further, since agents interactively collaborate in adaptive BPR, adaptivity at the MAS level is also realised. BPRs choose appropriate partners and pass messages with their corresponding content using the RR. Adaptivity of individual agents and the whole agent system is achieved through the different layers of our hierarchical knowledge model.

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 being able to be tailored and adapted for specific use due to their layered structure.

References

- Ambler, W.S. and Constantine, L.L. (2000) *The Unified Process Inception Phase*, Elsevier/CMP Books. **AUTHOR PLEASE SUPPLY LOCATION.**
- Arai, T. and Stolzenburg, F. (2002) 'Multiagent systems specification by UML statecharts aiming at intelligent manufacturing', *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pp.11–18. **AUTHOR PLEASE SUPPLY LOCATION.**

- Bauer, B., Muller, J.P. and Odell, J. (2001) 'Agent UML: a formalism for specifying multiagent software systems', *International Journal of Software Engineering and Knowledge Engineering*, Vol. 11, No. 3, pp.207–230. **AUTHOR PLEASE CITE THIS REFERENE IN TEXT.**
- Cossentino, M., Burrafato, P., Lombardo, S. and Sabatucci, L. (2002) 'Introducing pattern reuse in the design of multi-agent systems', *AITA'02 Workshop at NDe02*. **AUTHOR PLEASE SUPPLY LOCATION.**
- Dardenne, A., Lamsweerde, A. and Fickas, S. (1993) 'Goal-directed requirements acquisition', *Science of Computer Programming*, Vol. 20, Nos. 1–2, pp.3–50.
- Donzelli and Bresciani (2003) **AUTHOR PLEASE SUPPLY FULL DETAILS.**
- Dumas, M., Aalst, W. and Hofstede, A. (2005) *Process-Aware Information Systems: Bridging People and Software through Process Technology*, Wiley-Interscience. **AUTHOR PLEASE SUPPLY LOCATION.**
- Ehrler, L. and Cranefield, S. (2004) 'Executing agent uml diagrams', *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, pp.906–913. **AUTHOR PLEASE SUPPLY LOCATION.**
- Feather, M.S. (1994) 'Composite system design', *ICSE-16 Workshop on Research Issues in the Intersection Between Software Engineering and Artificial Intelligence, International Conference on Software Engineering*, Sorrento, Italy.
- FIPA Agent UML Website (2007) <http://www.auml.org/>, Accessed 1st March.
- Heymans and Dubois (1998) **AUTHOR PLEASE SUPPLY FULL DETAILS.**
- Jennings, N. (2000) 'On agent-based software engineering', *Artificial Intelligence*, Vol. 117, No. 2, pp.277–296.
- Jennings, N., Sycara, K. and Wooldridge, M. (1998) 'A roadmap of agent research and development', *Autonomous Agents and Multi-Agent Systems*, Vol. 1, pp.7–38.
- Laleci, G.B., Kabak, Y., Dogac, A., Cingil, I., Kirbas, S., Yildiz, A., Sinir, S., Ozdakis, O. and Ozturk, O. (2004) 'A platform for agent behavior design and multi agent orchestration', *Agent-Oriented Software Engineering Workshop, the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems*. **AUTHOR PLEASE SUPPLY LOCATION AND PAGE RANGE.**
- Lehman M. and Belady, L. (1985) *Program Evolution: Processes of Software Change*, Academic Press, London.
- Lotzsch *et al.* (2004) **AUTHOR PLEASE SUPPLY FULL DETAILS.**
- OMG (2006) *MDA Guide V1.0.1*, Object Management Group, Inc., Available from <http://www.omg.org/cgi-bin/doc?omg/03-06-01>, Accessed 1st March.
- Pollock, J. and Hodgson, R. (2004) *Adaptive Information: Improving Business Through Semantic Interoperability, Grid Computing, and Enterprise Integration*, Wiley-Interscience. **AUTHOR PLEASE SUPPLY LOCATION.**
- Schreiber, G., Wielinga, B., Hoog, R., Akkermans, H. and Velde, W. (1994) 'CommonKADS: a comprehensive methodology for kbs development', *IEEE Expert: Intelligent Systems and Their Applications*, Vol. 9, No. 6, pp.28–37.
- Shoham, Y. (1993) 'Agent-oriented programming', *Artificial Intelligence*, Vol. 60, pp.51–92.
- Sommerville, I. (2007) *Software Engineering*, 8th ed., Addison-Wesley. **AUTHOR PLEASE SUPPLY LOCATION.**
- Tveit, A. (2001) 'A survey of agent-oriented software engineering', *Proceedings of the First NTNU CSGS Conference, Norwegian University of Science and Technology*. **AUTHOR PLEASE SUPPLY LOCATION AND PAGE RANGE.**
- Wasserman, A. (1996) 'Toward a discipline of software engineering', *IEEE Software*, Vol. 13, No. 6, pp.23–31.
- Wooldridge, M. (1997) 'Agent-based software engineering', *IEE Proceedings on Software Engineering*, Vol. 144, No. 1, pp.26–37.
- Wooldridge, M. and Ciancarini, P. (2001) 'Agent-oriented software engineering: the state of the art', in Ciancarini, P. and Wooldridge, M. (Eds.): *Agent-Oriented Software Engineering: First International Workshop (AOSE'2000)*, LNCS 1957, Springer. **AUTHOR PLEASE SUPPLY LOCATION AND PAGE RANGE.**
- Wooldridge, M., Jennings, N. and Kinny, D. (1999) 'A methodology for agent-oriented analysis and design', *Proceedings of the Third International Conference on Autonomous Agents (Agents-99)*, Seattle, WA, p.28.
- Xiao, L. and Greer, D. (2005a) '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)*, pp.62–67. **AUTHOR PLEASE SUPPLY LOCATION.**
- Xiao, L. and Greer, D. (2005b) 'Modelling agent knowledge with business rules', *Proceedings of the Seventeenth International Conference on Software Engineering and Knowledge Engineering (SEKE'05)*, pp.566–571. **AUTHOR PLEASE SUPPLY LOCATION.**
- Xiao, L. and Greer, D. (2006) 'The agent-rule-class framework for multi-agent systems', *International Journal of Multi-Agent and Grid Computing*, Vol. 2, No. 4, pp.155–177.
- Yourdon, E., Whitehead, K., Thomann, J., Oppel, K. and Nevermann, P. (1995) *Mainstream Objects: An Analysis and Design Approach for Business*, Prentice-Hall. **AUTHOR PLEASE SUPPLY LOCATION.**
- Yu, E. (1997) 'Why agent-oriented requirements engineering', *Proceedings of the 3rd International Workshop on Requirements Engineering: Foundations for Software Quality*. **AUTHOR PLEASE SUPPLY LOCATION AND PAGE RANGE.**