Adaptation and Maintenance of Multi-agent Systems using Reaction Rules

Liang Xiao
Hubei University of Technology
Hubei, China

***Abstract*-Software systems must be adaptable and maintainable in a business environment to remain useful. Business rules are central to business requirements and their adaptability and maintainability is the key to keep systems up to the changing environment over time. In this paper, we propose reaction rules for modelling interactive agent behaviour in Multi-Agent Systems. A rule-based knowledge model is sourced from the business requirements specification and becomes a knowledge base of driving agent behaviour dynamically, at runtime. Because the knowledge model is externalised from the running system itself, system behaviour can be adapted and maintained in an easier manner. The work is illustrated using the case of a rail track management system.**

Keywords- business model, multi-agent system, reaction rule, software adaptability

I. BACKGROUND AND INTRODUCTION

In every organisation there are business rules, being compact statements that lay down what must or must not be the case in some aspect of a business [1]. According to the Object Management Group [2], business rules are "declarations of policy or conditions that must be satisfied". They often not only express policies, but also refer to administrative processes which determine how several tasks have to be executed [3]. A typical rule can be as simple as constraining a value among a list of valid values, and as complicated as to control a business process depending on the condition of some attribute(s) or a current policy. Most likely, several business rules may together constrain one aspect of the business, for example, the execution of a business process. They are characterised by their strategic importance to the business and deserve special attention [4].

Business rules capture requirements [5] including the decisions, guidelines and controls which make up the functionality [6]. As long as inconsistency and ambiguity are identified, either in rules or by rules, they are proved to be useful for capturing and resolving conflicts, both at the requirements level [7] and at the design level [8]. However, to date, rather than made explicit as part of the requirements document, their importance is ignored and often embedded directly into the final software product. This not only leads to the misinterpretation by developers with their own assumptions, but also provides no way to trace the rules in the code back to the requirements. In the end, the lack of explicit capturing of business rules contributes to rework and other inefficiencies [9].

Some suggest that business rules have the form of terms, facts, factor clauses and action clauses, and can be expressed in natural language using a tailored taxonomy [9]. According to this suggestion, if business rules are wrong, or subject to frequent change according to business needs, then there will be a lot of maintenance work. Thus, better still is the case where business rules are present in a structured natural language format and also executable by the system. Finding a way to make these rules executable would contribute significantly to the solution of transforming functional requirements to the final product.

It has not been agreed in the OO community where rules should be put in OO models. It is pointed out in [10] that the current situation where rules are spread out and converted into methods is a weakness of OO approaches. Although rules are used by the Unified Modelling Language (UML) and the Rational Unified Process [11] advocates them in presentation and conversation, sufficient guidance for business rules has never been provided [9]. The only practical representation of rules is by using the constraint element of Object Constraint Language (OCL) in the standard UML. However, this is at the design level as opposed to the requirements level where the interface is with business people. Another stated deficiency of OCL is its lack of rigor, which might lead to ambiguities [12]. Moreover, its elements are static, externally connected to the main models as an "add-on" [13] and in isolation in many cases. It is argued in [14] that what is important to business rules is that they are not presented as isolated elements, but linked to other elements comprising the enterprise model. To properly derive business rule requirements as business assets and let them remain valuable to stakeholders, they should be elicited and validated in collaboration with business customers during requirements analysis. Moreover, the use of business rules can compensate for the lack of capability in existing UML diagrams for capturing behavioural semantics. This paper presents an even better alternative construct of Reaction Rules in structuring software requirements in the agent context and being used as a first-class citizen, an argument advocated in [9].

II. Reaction Rule scheme

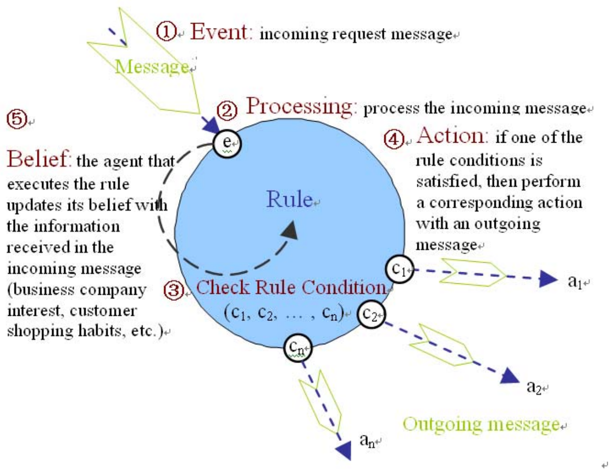

① Event: incoming request message
Message
② Processing: process the incoming message
⑤
Belief: the agent that executes the rule updates its belief with the information received in the incoming message (business company interest, customer shopping habits, etc.)
e
Rule
④ Action: if one of the rule conditions is satisfied, then perform a corresponding action with an outgoing message
③ Check Rule Condition (c1, c2, … , cn)
c1
c2
cn
a1
a2
an
Outgoing message

Figure 1. Reaction Rule scheme

In a Multi-Agent System, we regard agents as conceptual units for organising requirements and software units for realising responsibilities related with the relevant requirements. Requirements knowledge is modelled as rules to guide agent behaviour. An agent processes a rule using the following steps, as shown in Figure 1.

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

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

III. Reaction Rule Model

We are illustrating the application of our approach by the following case study.

Case Study (an extract of the British Railtrack system)
The system specification comprises several areas, including Train Running and Infrastructure Management. The Train Running domain supports the principal service to customers, including delivery of planned train paths and response to requests for changing train paths. The infrastructure of the railway system consists of the assets necessary to run the trains. Their condition is a major constraint on train running. Asset faults will cause train service re-planning. When a fault is detected, a Contractor responsible for the fault will carry out maintenance. When an Unauthorised Person (or even an Animal) is reported to get into track he or she may need to be protected by a track restriction, for example, an emergency isolation. An Unauthorised Person will access to the track implicitly and temporarily, while Contractor staff will require access to the faulty track explicitly and in a considerable duration. Both situations will involve the imposing of track restrictions by Infrastructure Management and the rescheduling of the affected train services by Train Running.

In our approach, we define an event-driven agent architecture, agents being responsive to events according to rules. Reaction Rules (RRs) 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 [15]. In many agent-oriented systems, agents monitor the occurrence of interesting events and respond to them [16]. 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

[17]. 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 2. *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 2, 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 2. Reconstructed function specification becomes a Reaction Rule

In the rule-based knowledge modelling perspective as stated in Section 2, 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.

IV. ADAPTATION OF MAS VIA REACTION RULES

We now illustrate how adaptability can be achieved through the approach by using agent IMI and IME from the case study.

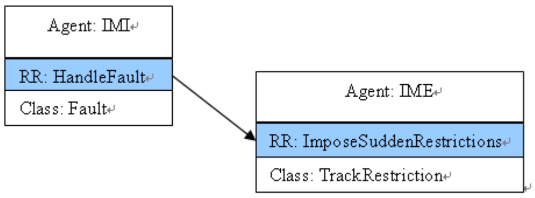
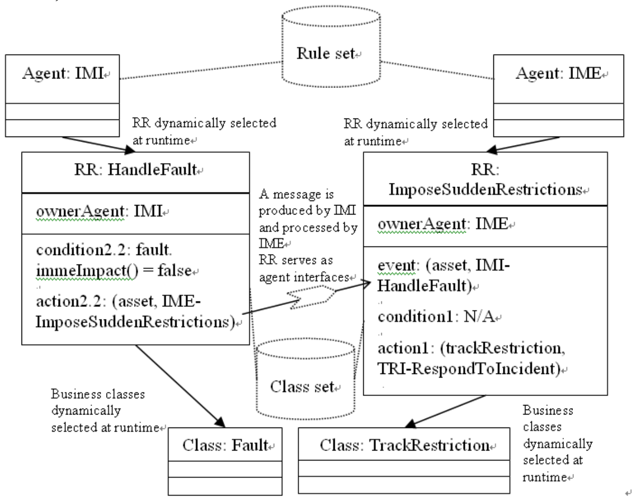


Figure 3. Agent, rule, and class

Figure 3 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 RR respectively in assistance of event processing and decision making. This scheme thus has a hierarchy where Agent-Rule-Class is established. This might cause the illusion that the relationships between agents, the selection of RR, and the selection of business classes are fixed.


Rule set
Agent: IMI
Agent: IME
RR dynamically selected at runtime
RR dynamically selected at runtime
RR: HandleFault
ownerAgent: IMI
condition2.2: fault.immeImpact() = false
action2.2: (asset, IME-ImposeSuddenRestrictions)
A message is produced by IMI and processed by IME
RR serves as agent interfaces
RR: ImposeSuddenRestrictions
ownerAgent: IME
event: (asset, IMI-HandleFault)
condition1: N/A
action1: (trackRestriction, TRI-RespondToIncident)
Class set
Business classes dynamically selected at runtime
Class: Fault
Class: TrackRestriction
Business classes dynamically selected at runtime
Figure 4. The actual relationships among agent, rule, class

In fact, there is no direct link between agents, or between agents and classes. Rather, such collaboration 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. 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 4. This shows the dynamic collaboration between agent-agent and agent-class.

Adaptability is thus demonstrated as achievable in different levels by reconfiguring business models. Not only agents, classes and methods are chosen dynamically, but also the link among these model elements are established dynamically, and interpreted by agents thereafter.

Connections among collaborative agents in business processes are decided at runtime and thus the enactment of business processes, as their 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, as conditions and corresponding actions have been re-built.

Figure 5 shows various aspects of adaptability that RRs can help to achieve, demonstrated through it compositional parts.

1. Partnership adaptability. In the figure, agent1 adapts its partnership from being between agent2 and agent3 as a new action is chosen. The goal of the business process is realised though the process is composed differently. A different outcome of the business process might be expected but the result serves the same aim of setting up the process in business.

2. PR application adaptability. The policy rules (PRs) that get applied are decided at runtime: a sequence of PR 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. In the same figure, agent1 adapts its PR sequence using PR2a as a replacement for PR2b as a result of the changed PR2a/b pre-conditions or PR1 post-condition.

3 & 4. Class application adaptability and method application adaptability. 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. In the same figure, agent1 adapts its use of ClassA1 instead of ClassA2, and a new method from Class B comes into use as a new corresponding business concept is accommodated. 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.

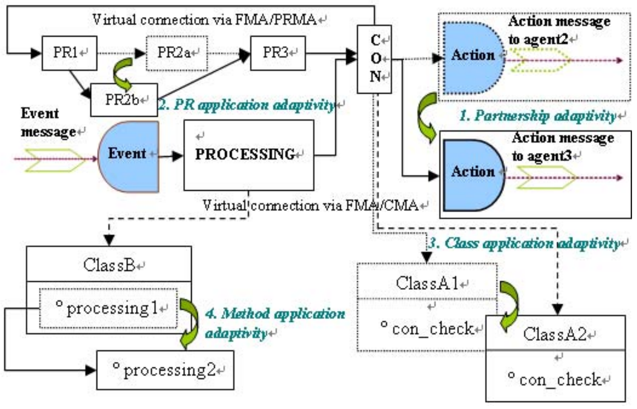

Virtual connection via FMA/PRMA
PR1
PR2a
PR3
PR2b
C O N
Action
Action message to agent2
2. PR application adaptivity
1. Partnership adaptivity
Event message
Event
PROCESSING
Action
Action message to agent3
Virtual connection via FMA/CMA
3. Class application adaptivity
ClassB
processing1
4. Method application adaptivity
processing2
ClassA1
con_check
ClassA2
con_check

Figure 5. Adaptability via Reaction Rules

V. CONCLUSIONS

The contribution of the approach is in using rules between agents and classes, so that, original classes are kept intact most of the time as stable components. Therefore, an easy way is provided to configure rules, which make use of classes differently according to the configuration. Agents are empowered to behave according to rules specification at run-time and, through these, flexible system behaviour is achieved. Overall, the potential for adaptability is promising especially since the benefits of the OO paradigm are not abandoned and an extra layer of agents introduced and coupled with knowledge in business rules. The approach will be made more powerful, but work so far indicates that it will contribute a novel approach to system adaptability.

REFERENCES

[1] Morgan, T., Business Rules and Information Systems, Addison-Wesley, 2002.

[2] Object Management Group, Inc., 250 First Ave. Suite 100, Needham, MA 02494, USA.

[3] Korthaus, A., "Using UML for business object based systems modelling", In M. Schader and A. Korthaus, editors, The Unified Modeling Language - Technical Aspects and Applications, pp. 220-237, Physica-Verlag, Heidelberg, Germany, 1998.

[4] Rosca, D., Feblowitz, M. & Wild, C., "Decision Making Methodology in Support of the Business Rules Lifecycle", Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97), p.236, January 05-08, 1997

[5] Rai, V.K. & Anantaram, C., "Structuring business rules interactions", Electronic Commerce Research and Applications 3(1): 54-73, 2004.

[6] Kleppe, A., Warmer, J. & Bast, W., MDA Explained: The Model Driven Architecture: Practice and Promise, Addison-Wesley, 2003.

[7] Rosca, D. & Wild, C., "Towards a flexible deployment of business rules", Expert Systems with Applications 23(4): 385-394, 2002.

[8] Liu, W., Easterbrook, M.S. & Mylopoulos, J., "Rule-based detection of inconsistency in UML models", Workshop on Consistency Problems in UML-Based Software Development, 5th International Conference on the Unified Modeling Language, Dresden, Germany, 2002.

[9] Ambler, W.S. & Constantine, L.L., The Unified Process Inception Phase: Best Practices in Implementing the UP, R&D, 2000.

[10] Nilsson, B.E., "On why to model what and how: concepts and architecture for change", in: Nilsson, A.G. et al. (Ed.), Perspectives on Business Modelling—Understanding and Changing Organisations, Springer, New York, 1999.

[11] Kruchten, P., The Rational Unified Process: an Introduction, second edition, Addison-Wesley, Reading, MA, 2000.

[12] Korthaus, A., "Using UML for business object based systems modelling", In M. Schader and A. Korthaus, editors, The Unified Modeling Language - Technical Aspects and Applications, pp. 220-237, Physica-Verlag, Heidelberg, Germany, 1998.

[13] Warmer, J. & Kleppe, A., The Object Constraint Language: Getting Your Models Ready for MDA, Second Edition, Addison-Wesley, 2003.

[14] Bajec, M. & Krisper, M., "A methodology and tool support for managing business rules in organizations", Information Systems 30(6): 423-443, 2005.

[15] Wasserman, A.I., "Toward a Discipline of Software Engineering", IEEE Software 13(6): 23-31, 1996.

[16] Jennings, N.R., Sycara, K. & Wooldridge, M., "A Roadmap of Agent Research and Development", Autonomous Agents and Multi-Agent Systems 1(1): 7-38, 1998.

[17] Wooldridge, M., Jennings, N.R. & Kinny, D., "A Methodology for Agent-Oriented Analysis and Design", Proceedings of the 3rd International Conference on Autonomous Agents (Agents-99), pp. 69 - 76, 1999.