

# An adaptive security model using agent-oriented MDA

Liang Xiao \*

School of Electronics & Computer Science, University of Southampton, Southampton, SO17 1BJ, UK

## ARTICLE INFO

### Article history:

Available online 14 May 2008

### Keywords:

Adaptive security model  
Agent-oriented model-driven architecture  
Agent-oriented software engineering  
Role-based access control  
Adaptive and secure multi-agent systems

## ABSTRACT

Model-driven architecture (MDA) supports model-centred software development via successive model transformation. In MDA, the reusability of models is improved as well as the traceability of requirements. Agent-oriented model-driven architecture (AMDA) associates adaptive agents with a business-oriented interaction model and lets agents dynamically interpret their behaviour from the continuously maintained model via which the current business needs are deployed at runtime. The continuous re-interpretation rather than discrete re-transformation of models means immediate requirements deployment after re-configuration, no system down time being required to affect changes and results in a development process that is oriented to business experts rather than developers. Adopting the adaptive agent model, an AMDA paradigm, we put forward a security-aware model-driven mechanism by using an extension of the role-based access control (RBAC) model. For this purpose, the concept of agent role proposed in agent-oriented software engineering (AOSE) is integrated with the one proposed in RBAC. Agent duties are specified in an interaction model and describe the roles that agents can play to fulfil their functional responsibilities. Agent rights are specified in a security policy rule model attached to the interaction model and describe constraints upon agent capabilities caused by their associated social roles. The role-based interaction and policy-driven model incorporates both agent rights and duties. Hence, functional requirements and non-functional security constraint requirements are put together, related by the concept of role. Consequently, agents can continuously use the re-configurable model to play their roles in order to fulfil their responsibilities, and at the same time respect the security constraints. The major contribution from the approach is a method for building adaptive and secure MAS, following model-driven architecture. The approach is illustrated with an actual British railway management system.

© 2008 Elsevier B.V. All rights reserved.

## 1. Introduction

Model based development for secure systems can benefit from high level abstraction and model transformation in a model-centric development paradigm, at the same time providing improved productivity and platform independence. Security, as part of the system requirements, must be integrated into the system model from the very beginning, if full advantage of model-driven software development is to be taken. However, existing approaches using model-centric development typically ignore security at the modelling stage, rather adding security aspects later in an ad-hoc manner. This can lead to deficiencies in security and more importantly, a less cost-effective development process. The fact that security requirements are usually not considered as an essential part of the overall system requirements is reflected in the fact that design models often describe only what components in a system *should do*, as in traditional functional requirements. What is

equally important is what the systems *should not do*. Malicious misuse/abuse should be prohibited in the non-functional requirements of security but, unfortunately, the latter is sometimes ignored or considered very late in the system development process. The separation of production and protection of systems results in vulnerability and a lack of integration.

Addressing security separately late in development also leads to poor maintainability. This is due to the complex and cross-cutting nature of the security requirements. Late discovery of security requirements will lead to costly correction or adaptation, sometimes after software development has been completed. Without at least abstracting and preferably decoupling the security requirements, the development and maintenance of secure software systems will remain a significant challenge. A means of modelling security requirements as part of the system model and adapting them accordingly afterwards must be provided. One possible approach is to use a model transformation approach, where the model includes security needs and where these are traceable and easily adapted later.

The object management group's (OMG) model-driven architecture (MDA) paradigm [8,9] is now well established. The paradigm

\* Tel.: +44 (0) 23 8059 3269.

E-mail address: [lx@ecs.soton.ac.uk](mailto:lx@ecs.soton.ac.uk)

has been advocated for productive software development, based on successive model transformation. Models are built for reuse and once re-configured, systems can be re-generated. Hence, code change is minimised when requirements change. Systems with mutable security needs can take advantages of the paradigm if security is explicitly modelled as part of the system model at a suitable level of abstraction. Adaptivity and maintainability of systems can then be enhanced since changes to the model, including functional requirements as well as non-functional security requirements, will lead to automatic behavioural change in running systems.

The adaptive agent model (AAM) [27,28,32] allows an agent-oriented model-driven architecture (AMDA) [29] paradigm, compatible with MDA and appropriate for multi-agent systems (MAS). AAM combines the dynamic features of agent with the advanced development paradigm of MDA. Briefly, AAM is a methodology that guides the building of an organised hierarchical business knowledge model [31] to drive adaptive agent system behaviour. The model originates from business requirements [35], is interpreted and executed dynamically by agents at runtime, and is under continuous maintenance by business people. Tools [30] have been developed to support the documentation and maintenance of the model. Existing components and services can be reused to support agents to execute business requirements captured in the model [33,34]. A model-driven security model as integrated in AAM is put forward in this paper. We consider the very important access control issue as our security focus, formalising a security model centred on policy rules. Agents make use of the integrated model to perform functional capabilities, constrained by the security policies. Business experts and decision makers can continuously maintain the integrated model in order to reflect changing business needs.

An agent-oriented MDA solution is appropriate since agents residing within the MAS under development can conceptually have security constraints while also realising their functional behaviour. In this way, security requirements are integrated with functional requirements. agent-oriented software engineering (AOSE) considers system model building to be centred on a role concept, role representing functional requirements. Role-based access control (RBAC) considers the security model to be centred on role, role representing non-functional security requirements. Unifying these role concepts offers the fundamental core model element on which an integrated model can be built. A new combined role notion can be used to pull duties and rights together in a single model. In this way, security requirements become first class citizens along with functional requirements and can also start at the requirements modelling phase.

The rest of the paper is structured as follows. The next section will investigate the existing modelling approaches towards security, the current status of security development in MAS, and in particular, the role concept as in AOSE and in RBAC. In Section 3, we describe our approach in detail, including an overview of the proposed approach, an illustration of the original AAM interaction model, the security model add-on, and their integration using a unified role notion. Section 4 discusses a British railway management system as our case study. Our approach will be applied and a CIM, PIM, as well as a PSM and code be developed, using an agent-oriented MDA paradigm. Section 5 demonstrates the adaptivity that has been achieved and evaluates the advantages of the new approach over traditional methods. Finally, Section 6 discusses the contribution this work makes to the state of the art, in the areas of agent-oriented software engineering, model-driven architecture, distributed computing, and role-based access control, as well as provides some conclusions to the paper.

## 2. Background

### 2.1. Security modelling

UML does not explicitly provide security modelling. However, one may extend the standard language to accommodate extra concepts. Defining a profile is one extension mechanism that can be used for this purpose. In [6] an extension to the business process modelling notation (BPMN) has been described, security requirements being incorporated as part of a UML profile that supports security modelling in business process diagrams. In a related approach [7,42] a UML profile for secure data warehouses has been developed. In this approach OCL is adapted for expressing secure constraint rules. Both approaches offer some support to describe security concepts in the overall modelling process but the languages alone are insufficient if they are not part of a complete solution for developing secure software systems in practice (the transformation of security constraint rules to system implementation).

Stereotypes have been used extensively in UMLsec with an aim of expressing security constraints that must be satisfied in the design level [5]. The modelling method does provide a means to embed security requirements in UML models and in doing so supports security analysis against models [1]. For example, a secure communication link might be expressed as necessary between a buyer and a seller, ensuring a fair transaction after which the customer either receives the goods purchased or is able to reclaim the payment in the case of an unsuccessful trade. Going even further, a framework towards secure system development that combines the Tropos methodology and the UMLsec modelling method has been proposed in [2] to support a modelling process covering requirements analysis to design model building. This moves the modelling language forward to a model-driven security architecture. However, it also has a number of weaknesses. First of all, its OO based permission model is constrained in the types of resources to be protected and actions to be performed, just like some other approaches. Generally, permissions are specified saying which (active) object can perform what restricted actions upon which other (passive) objects (files, classes, etc.). Security communication protocols are modelled based on these. The application of such secure access is limited to protecting file systems (actions: read, write, etc.), programmed object archives (actions: class method invocation, etc.), network resources, and similarly other low level system resources. Furthermore, the approach focuses on transportation-level communication where secure keys are used among components under protection and so only at that level the security constraints are enforced. No mechanism has been offered for differentiating various levels of access rights and their assignment to subjects (e.g., according to organisational positions) for the access of various resources, the type of which should also being in a wider range. More importantly and related to this, the modelling is not at a business level with regard to critical business service protection. Business experts may find it difficult to understand or configure secure access policies according to their business needs.

Modelling approaches would be more useful if an adaptive business-level access control model were in place. Aiming at building such a model at a high level of abstraction, agents may be an appropriate construct to abstract access subjects. The interaction among collaborative agents, which in turn involves individual internal computation and critical system resource access, is the means system goals are achieved and services delivered to the users. The use of system resources and the delivery of business services in such interaction processes, therefore, must be controlled to ensure the achievement of goals is not compromised by malicious acts. Thus, a security-aware interaction process must be modelled,

capturing the knowledge to be used by agents of the conditions for performing the requested tasks as well as doing them properly. Ultimately, agents dynamically execute the model through which services are composed and provided according to security needs.

## 2.2. Security in MAS

Arguably, the use of agent in security engineering is still in its infancy. To compound matters, security concerns in the MAS domain have not been emphasised in a scale compatible with its widespread academic research and industry application. More forcefully put, agent systems themselves are not typically secure. Currently, MAS platforms and standards organisations for MAS offer a security mechanism at a very low level. FIPA [10] uses an agent platform security manager [11] for secure communication. Security related parameters can be specified in an agent message envelope, indicating its requirement for security service, according to the FIPA ACL message standard. The agent platform security manager and the agent management system can then be used in agent platforms to maintain security via public and private key authentication or public key infrastructure. However, this provides secure agent communication at the transportation level only. The lack of security support by FIPA as well as the FIPA compliant Java Agent Development (JADE) platform [12] has been identified in [13–15]. The JADE-S [17] add-on of the JADE platform provides some degree of security support. Its potential usefulness lies mainly in two aspects: secure agent message transportation using message signature and encryption; and an access control mechanism based on JAAS (Java Authentication and Authorization Service). Again this add-on offers only transportation-level security. Its use of policy files is intended only for general-purpose access control to local resources such as system files, network, and so on. Permissions are defined on the basis of system-oriented actions such as: passing messages, moving among containers, and creating/killing agents. Apart from work at the platform level, some attempts have been made to address the security issues of mobile agents, which are vulnerable to threats due to their mobility [16]. Unfortunately, no general mechanism has been offered to cope with the control of agent access rights to resources such as critical data and services, in the context of their running on behalf of human beings or systems.

Therefore, the responsibility of enforcing security in MAS goes to the agent designers and developers who have to meet particular business-specific needs. Usually distributed over their runtime environment, agents behave and interact with each other dynamically, if not fully autonomously. Access control in such a highly dynamic environment is very difficult due to the large number of agent interactions, unforeseen agent behaviour, and agent interaction process that could emerge at runtime. The agent research community and developers largely ignore the security issue given the complexity of implementing MAS-specific security requirements on top of existing functional requirements. Ignorance of security constraints would let the potential autonomous agent behaviour impact negatively upon systems, such behaviour not being predicted in their original design. Imposing the right control upon agent behaviour, on the contrary, can provide authorised system access and resource consumption. Overall, the dynamic feature of agents leads to the fact that they are not always associated with a fixed set of data, but rather intentionally interact with each other and human beings flexibly, in the process of which they may have access to various system data and services. This brings unique challenges and a changed view of security modelling that we must bear in mind.

Security models such as the RBAC model are often concerned about permission control for human users. In MAS, some agents operate on behalf of users and others perform business functions

as a result of system design. The concepts of agent rights and agent roles, resembling those of user rights and user roles are absent in the context of system resource access. This imposes difficulty on general modelling of all access subjects when access control is concerned. Both human actors and system agents should be restricted in their behaviour in a MAS as a result of security needs.

Apart from the access subjects, the access objects need matching notions in the design of a unique MAS security model. The agent concept in the resource access pattern raises the level of abstraction of access subjects. The usual access objects of types of files, objects, and class methods in the OO paradigm, however, do not fit well in the pattern. High level business services that can be used, managed, or composed by agents are sometimes believed at the appropriate level of abstraction of agents. Actually, building agents on top on web services has been advocated in [18] as an essential way to produce real business value from the agent paradigm. In doing so, agents fulfil system services. We can, therefore, regard agents as the agents of services through which services are accessed and made use of. But work in the direction of secure service access management via agents is rare.

Since simply borrowing OO security mechanisms does not work and little work has been done in security modelling in MAS, there is a need to investigate the mainstream security access control models in order to discover a useful one and adapt/extend it as required to fit the MAS context. The alternative of designing a brand new security model applicable only to the MAS domain involves reinventing and may impose a barrier to the existing security engineering community. This approach is analogous to the origin of Agent UML (AUMML), which is an extension of UML for modelling MAS [22]. The next section will introduce such a security model that we can adapt for reuse in MAS towards AMDA.

## 2.3. Role access control in RBAC and role playing in AOSE

Access control is central to the security of software systems. One access control model that supports efficient management is the widely accepted US National Institute of Standards and Technology model of role-based access control (RBAC) [19]. In RBAC, as illustrated in Fig. 1, roles represent job functions in an organisation. They bring together users and permissions. Permissions that describe operations upon resources are associated with roles. Users are assigned to roles to gain permissions that allow them to perform particular job functions. A major benefit of using this type of model is that the reconfiguration of user–role, role–permission, and role–role relationships, directed by administrators, can reflect changing organisational policies. The maintenance of such a sub-system that is independent from the core application means minimal impact on the overall system of requirements changes with regard to security. RBAC is widely accepted as a best practice and implemented in one form or another in systems including Microsoft Active Directory, SELinux, FreeBSD, Solaris, and Oracle DBMS. However, several weaknesses have been identified in [33]. Major insufficiencies include: its incapability to assign different permis-

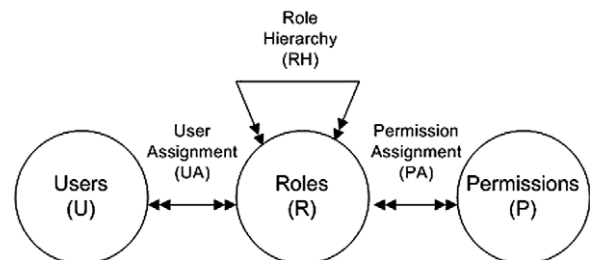


Fig. 1. The RBAC model.

sions to individuals with the same role; to differentiate users from geographically or socially different organisations and assign permissions accordingly; to allow in exceptional conditions, special permission assignment; to specify negative permissions (hence it is inconvenient to grant permissions to a group of users except particular individuals from the group).

The lack of mapping from a genuine organisational structure into a role structure using RBAC has been realised, leading to the development of the Enterprise RBAC model [36]. Enterprise roles naturally reflect the actual organisational roles required by the system. Users can be granted access rights across multiple systems, called the “target systems”, of an organisation. Enterprise Roles collect all permissions necessary to perform the specific roles, from the target systems. A general role hierarchy is supported.

Work on incorporating RBAC policies into UML design models towards modelling secure system has been reported in [3]. A key concept is the development of an application-specific model which is security unrelated and a context-specific RBAC template model and then merging them to obtain a security-aware compositional model. Such a merging process is not always straightforward, especially when concepts in the system domain cannot be mapped to RBAC concepts. Moreover, the use of an object-oriented template model determines its poor applicability to the MAS paradigm.

Another security modelling language based on RBAC and aimed at MDA is SecureUML [4]. However, it generates an infrastructure that is only concerned about the access control of the systems. Little attention has been paid to integrating such architecture with the rest of the systems. This attributes to the method the typical modelling problem seen in current software development process. Capturing not only permission constraints but behaviour duties in a single integrated model is necessary to avoid poor maintainability.

Role is also an important concept in agent-oriented software engineering (AOSE), agents usually being linked together by their role playing behaviour. Agent UML (AUML) [22] language uses role as a primary notion for MAS modelling. Role is also a principle concept in the Gaia [21] methodology, giving agents well-defined positions in the organisation and the associated behaviour that can be expected. Moreover, a meta-model is proposed in [20] using the notions of agent, role, and group to enable a three-way association, in which an agent can be assigned to playing a role within a group, gaining the specific task performing capability. One particular agent can play different roles in different groups, where what it is expected to perform depends on the context of the group. One group can be dynamically formed by agents that collectively have the capabilities to accomplish certain emergent tasks. This offers an organisational structure that may be adapted for use in a model that captures agent role playing behaviour in various interaction processes.

The role concept is of significant importance to both the AOSE research community and the RBAC community. However, the direction of work based on the role concept in them is completely distinct and no research has ever been carried out to reconcile the two concept definitions in MAS for security modelling. Since RBAC has no concept of duty and AOSE has no permission constraint for agents, the complementary nature allows integration of them in a single role-based interaction model. Our hypothesis is that, a unified role concept will support building systems where their architecture is driven by functional requirements and security requirements together. This requires the development of an appropriate security model that constrains agent behaviour based on roles, being integrated with a functional agent interaction/computation model that describes the routine agent role playing behaviour.

### 3. The model-based development approach for secure systems

#### 3.1. Approach overview

Our framework consists of: (1) an agent runtime platform; (2) agent instances running on the platform that are empowered by an engine with model interpretation capabilities; (3) a reaction rule model that the agent runtime engine can use to interpret reactive behaviour; (4) a policy rule model that the agent runtime engine can interpret global strategic constraints; (5) the overall model produced using (3) and (4) which agents use as their interaction and computation pattern running in business processes. Here, we focus upon the design of a modelling approach towards security-enabled model-driven architecture. Tools have been developed [27,30,32] that support the configuration of reaction rules and policy rules for human experts. These are stored in XML repositories and are interpreted for agents by the engine, the implementation of which is part of our future work.

The artefacts produced in the framework are mainly reaction rules and (security) policy rules which together are termed *interaction models* [29,32,34], capturing overall business processes. Apart from the combined platform-independent models (PIM), the system consists of agents running on any given platform and components or services that facilitate agent performance as advised by the models. The selection of the specific agent platforms triggers the transformation of PIM to platform-specific models (PSM) and code that are suitable to the running agent instances.

Building these artefacts and fitting them into the MDA paradigm shapes a parallel software development procedure in three directions.

(1) The development of an interaction model and security model (the knowledge models part in Fig. 2). The notations used in the modelling process will be agent-oriented UML (shown in Table 1). At the centre of the models are the rule elements, the semantics of which are further enriched to support a model-driven architecture. *Reaction rules* collectively fit agents into overall interaction patterns that capture the blueprint of the system. They individually specify the computation pattern of corresponding agents required by the interaction. *Policy rules* define the global constraints that must be satisfied in interaction or computation. The rule specifications that capture behavioural meanings or strategic constraints are part of the PIM and are in compliance with two types of rule schemes. These schemes conform to pre-agreed formalisms. Rules are uniformly specified according to the schemes which define their associated XML annotations in the instance level, when such models are specified. A model interpretation engine being capable of parsing and transforming them in a pre-determined manner, agents will be driven in the architecture under construction to present their unified behaviour of event handling, decision making, action performing, as well as constraint matching. This mechanism turns a conventional MDA's code generation process into a model interpretation process but keeps the same model-driven principle.

(2) The development of components/services for invocation by agents to assist their functions (the Classes part in Fig. 2). These components may have been developed for other purposes and re-used in the current context. They may be programmed in disparate languages but uniformly invoked via interfaces with the agents sitting in the above level.

(3) The development of agents for running the models and using the components and services (the agents part in Fig. 2).

We start the discussion of our approach from generically developing platform-independent models, coupled with abstract agent and component notions. Afterwards in the case study, we demonstrate how these models become platform-specific, coupled with agents and components running on the chosen platforms.



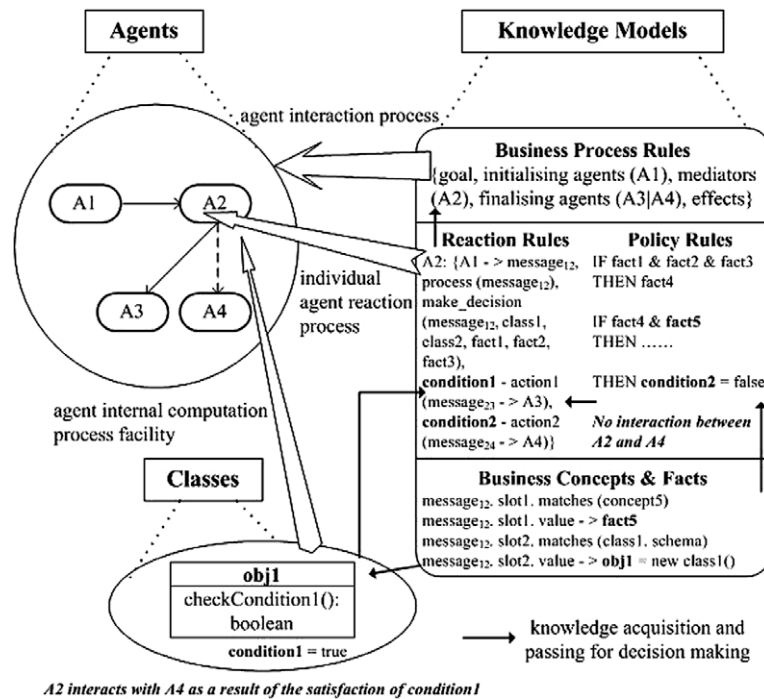




Fig. 2. The approach overview demonstrating the models and the agency.

Table 1

Notions and notations of the AAM

Interaction model		A protocol model that describes the interaction process of multiple agents aimed at a common goal One example here is an interaction model describing the interaction among several entities employed by a railway management system
Agent		A high level abstraction that conceptually has common goals shared with other agents and computationally has responsibilities for contributing to the goals. Rules are defined that decide the roles an agent should play in a certain interaction, aimed at a certain goal and the low level computational units should be invoked in the process. Object components, Web Services, and even other agents can all be used by an agent. Agents interact with one another by passing messages, the processing and producing of which is also determined by rules
Reaction rule		One example here is an "Infrastructure Management Incident" (IMI) agent that handles the incidents occurring to rail tracks and other assets A requirement capture unit that externalises agent knowledge and is configurable at runtime by domain experts. Agents use rules to understand and respond to messages, make decisions, and collaborate with each other. A collection of rules compose and define the agent interaction model. Rules can be defined for a single agent to play different roles in different interactions models
Component and service		IMI agent may use the reaction rule of "Handle Fault" to guide its interaction and computation behaviour, in a Railtrack interaction model Traditional passive components that respond to active agents when they are invoked, assisting the running agents to behave. The invocation of these low level units is defined in rules A component called "Fault" and a service called "Restrictions" may facilitate agent IMI to process events or other internal computation, during its execution of "Handle Fault" or other reaction rules
Message		An information container passing between agents. Messages are known by agents that create them and are expected by agents that receive them, if related rules are defined. String, XML fraction or even objects can be encoded in them. The passing of a message indicates that the sender has made its contribution towards a business goal and now the receiver takes its responsibility to contribute to the same overall goal A "track restriction request" message may be delivered by agent IMI, as a consequence of its execution of "Handle Fault", to other agents

### 3.2. AAM interaction model: reaction rule and policy rule

The adaptive agent model has been developed for large business applications to cope with changing business requirements and to ease the continuous maintenance of supporting software. AAM breaks the tradition of tightly coupled component/service relationships and provides a dynamic and on-demand invocation manner by using adaptive agents as a high level abstraction. The AAM approach was intended to add adaptivity to large object-oriented software systems. Nevertheless, when it is adapted for use in a distributed environment, agents as a high level abstraction can make use of a combination of existing components and services distributed heterogeneously to meet their required functionalities. In a heterogeneous environment, numerous constructs of a variety of types (objects, services, and agents) are available via independent development. Alternatives from competitive service providers can be explored for (re)use towards evolving business purposes. Apart from (i) component/service/agent providers and (ii) end users, a third important role might be played in such a system by (iii) interaction model designers. They have application domain knowledge, choose components and services from alternatives, and specify their interactions to fulfil business goals.

Although standard languages exist for describing component interfaces or defining interactive message passing protocols, for instance, DAML-S for Web Services and FIPA ACL and QML for MAS, their usage is limited in their specific areas. So far no mechanism for coordinating and interoperating disparate components has been provided. The difficulty becomes more severe if selecting the right components/services and specifying cooperation towards an emerging business goal must be accomplished in a timely way and dynamically. With AAM, we aim at reusing the already established software paradigms in our extended and integrated model-driven paradigm so that their interaction becomes automatic once the model has been specified, and the change to the interaction of runtime components/services only involves the change to the interaction model specification [34]. In this way, the potential weakness of MDA that would be exposed when it is applied to a heterogeneous environment and for coordination of separately developed components/services can be addressed by the agent-MDA paradigm of the AAM.

At the heart of the AAM is the interaction model that describes the interaction of a collection of agents towards a common goal. The interaction model is comprised of multiple reaction rules (RRs), each guiding an agent in its individual responsibility for service provision in reaction to other agents. This allows the agent to contribute to the overall goal. One or several policy rules (PRs) may be collectively selected and applied in such processes to reflect global business policies that must be complied in service fulfilment, so shared by all agents in the system.

The AAM framework built upon the notions and notations outlined in Table 1 can be described as having a three-layered architecture. The first layer consists of agents interacting with one another by passing messages, using the behavioural knowledge from the next layer. The middle layer is a structured knowledge-base of rules, supplying to agents from the previous layer and referring to the components and services from the next layer. The last layer consists of computational units, ready to be invoked to facilitate the execution of the interaction model. The knowledge in the middle layer is expected to be updated continuously during the running of the system corresponding to changing requirements at runtime.

A major construct being in the framework, reaction rules specify interactive agent behaviour. A simplified graphic form of RR scheme is shown as in Fig. 3 describing the message passing among agents and the use of components/services during message processing and producing.

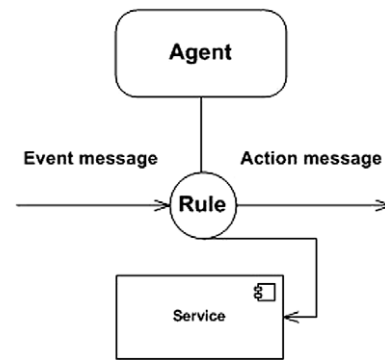


Fig. 3. The basic reaction rule scheme (decision making process ignored).

Although this scheme gives an impression that the approach uses rules to bind up certain agents in the interaction model, the system architecture is actually dynamically formed and driven by a collection of rules selected and appropriately fitted into the particular context at each single runtime occasion. A complete RR specification can be formalised as the following: on receipt of an event message, processing the message, and then making a decision which results in performing proper actions associated with the current conditions, and after that updating the belief corresponding to the incoming event, if appropriate. Reaction rules define agreements that are bound between agents for their interactions, constraining what and how agents should perform in a reactive and proactive manner in business processes. Driven by events, agents use RRs to make business decisions. Fig. 4 represents the schematic decision making tree. Each RR's decision making element can be decomposed into multiple {condition, action} couplets, actions performed following conditions evaluated as satisfactory on the selected tree branch while making the decision. A branch such as (Con2 → Con2.2) may be selected if these both conditions are met but neither condition Con2.2.1 nor Con2.2.2 is satisfied in the branch further down. If additional conditions need to be considered, the selected tree branch may be extended with: {condition2.2.3, action2.2.3}, {condition2.2.3.1, action2.2.3.1}, and so on, which could be found as required when new needs come into the system.

Based on the basic RR structure and its decision making tree structure, a RR scheme is defined as:

**{event, processing, {condition, action}<sub>n</sub>, belief}**

When an event message is received by an agent (Step 1), information contained within (business objects, etc.) is decoded from it and facts are then known to the recipient (Step 2). To respond, the agent makes a decision and performs actions that are associated with the satisfied conditions (Step 3). The result of the actions could be producing event messages to other agents (Step 4). The agent's beliefs are updated with the new information (Step 5).

Roles that agents shall play are actually specified implicitly inside reaction rules. Indeed, to any agent, the execution of a RR represents the process of playing a role. However, the exchangeable

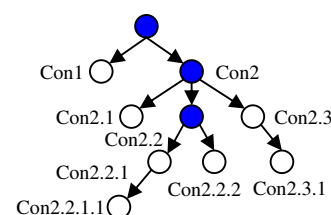


Fig. 4. Schematic decision making tree of a RR.

use of rule and role is conceptually inappropriate. It is possible and sometimes necessary to define a rule for an agent that configures one role in one interaction model and, another role in another interaction model. This corresponds to situations in reality. For instance, a person at work plays a different role from the role he/she plays at home. A rule must be capable of taking into account the different contexts in which multiple roles are played and its form should not be subject to change when the context changes. The agent-role-group meta-model [20] proposes to organise agents' role playing behaviour in groups. However, this static structuring method is not the most rationale and flexible. The use of the RR construct in the interaction model, on the contrary, naturally directs agents to play various roles in various collaborative opportunities towards various goals. The re-configurability of RRs is, therefore, of significant importance to adding adaptivity to reaction/interaction-intensive systems. In the scenario given above, a person usually processes a task request from his/her boss/colleagues in workplace differently from a request from a family member. The distinct role playing scenarios of a single entity in distinct interactions can be defined in a single rule (practically easy by specifying multiple condition and action pairs), or several for structuring purposes. The distinction of rule concept and role concept is necessary and useful, rules being abstracted as a layer for configuration purposes, above the role level which describes either functional duties or social rights. The latter type of role is related with policy rule, in contrast with the former being related with reaction rule.

Business and organisation regulations and strategies usually should override individual behaviour as a need of efficient management, giving a sense of global control and shared value that everyone should respect. To this end, the policy rules complementary to reaction rules come into play, sometimes applied to constrain or facilitate individual agent decision making. These often take the form of "IF...THEN...". They can be used to impose extra condition statements upon executing RR, and entail that business entities must meet specific logical relationships or within a particular range of values as a requirement of business strategies. The consequence of the application of a PR might be the production of business facts which should be respected and such knowledge could guide the direction of behaviour. A chain of PRs may be formed and applied iteratively, as a result of the application of one PR accumulating additional knowledge that triggers other PRs to function. The use of many RRs and PRs eventually formulate

the blueprint of the overall system model, capturing system requirements and achieving system goals. Fig. 5 illustrates a portion of an interaction model, involving two RRs (owned by two agents) and four PRs (shared among all agents). RR1 and RR2 are used by Agent 1 and Agent 2 in an interaction. When RR1 receives an event, it invokes some shared services for the processing. Then several PRs are applied due to their satisfied conditions. This leads to the production of some facts that facilitate decision making. An action is finally selected among available execution paths and triggers RR2 to function in Agent 2.

An important reason why policy rules are distinct from reaction rules is that the same concern may be spread over the system and many entities may have to adhere to each of these globally applicable concerns. Likewise, a wide range of system resources accessible to each subject may share common access strategies. If such a strategy changes, then in order to comply with such changes we have to change the configuration of many places where the subject has access to resources. In particular, changes go to many RRs through which agents manage and provide services and data. Externalisation of PRs and later security specific PRs is therefore crucial to a security-manageable system. The embedment of PRs into RRs could inevitably put a heavy burden upon system maintenance. However, the general structure that generic PRs take should be adapted for security PRs. This is again for better policy management and maintenance. For example, employees and managers may have different access levels to resources for good reasons. Affiliate organisations may have more access rights to resources than the others. Categorising subjects by roles and organisations simplifies the management matter, one rule being designed and then applicable for a group of subjects. At the same time, it is also very likely that some individuals need superior permissions for particular resources. This disallows us to exclude the definition of policies on an individual basis. To conclude, when security PRs are formulated, the form should uniformly take into account all potential factors, so that their number is minimised, management made easier, and dynamic binding with RRs kept efficient.

Combining such versatile security PRs and RRs with the existing AAM framework offers many benefits. In particular, we can make use of its adaptive properties [27,28,32] and model-driven features [29]. The RR type of rules shown in Fig. 5 dictates functional duties that agents must perform to fulfil its capabilities and contribute to common a goal. The PR type of rules enforces some kind of social regulations that could be applicable to the whole business organi-

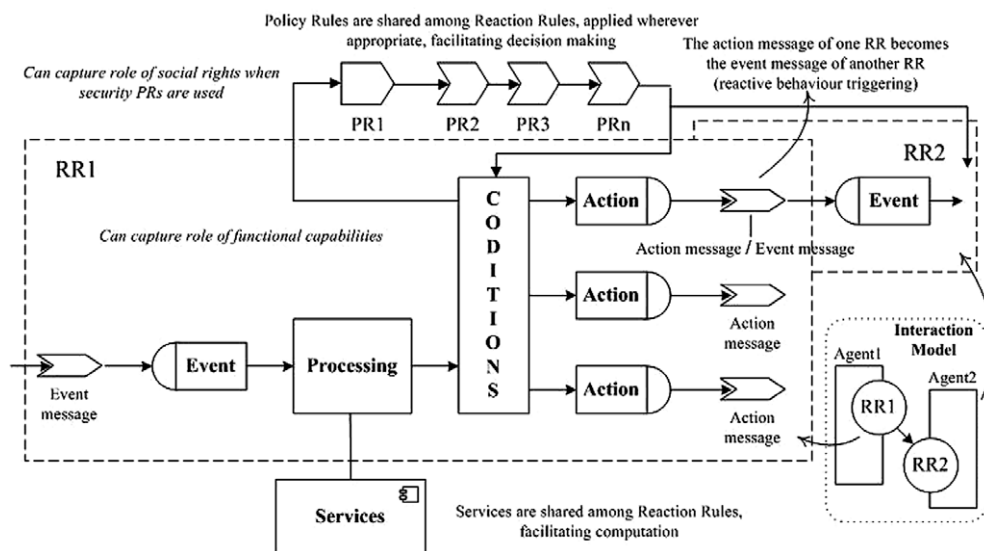


Fig. 5. RR and PR work together in an interaction model.

sation before such performance takes place. In the view of an agent's functional role playing and social role associated with permission, these two types of rules can fully capture both functional needs and non-functional security needs in an integrated interaction model for secure agent behaviour modelling. That is, before a role of functional capabilities is performed, its role of social rights must be checked against that performance. This adaptive and integrated RR and PR scheme is built upon the integrated role notion. The applicable security-PRs, which are externally specified, are dynamically bound with RRs so that the role of social rights is bound with the role of functional duties. Only in this way is the role concept completed and the role playing guaranteed to start if and only if it has appropriately granted permissions. The next section illustrates security PRs formalised in a structure suitable for easy organisation and maintenance and at the same time fully integral with RRs in the existing AAM framework.

### 3.3. AAM security meta-model and security-PR: an extension towards integration

As described in the previous sections, policy rules can be enforced while reaction rules are performing. AAM was originally intended to use policy rules to capture domain-independent, general-purpose policy requirements. When PR use is extended to enforce security policies, PRs can be applied within the existing framework straightforwardly since they comply with the generalised form. However, being concerned with security, the formalism needs to capture the specific knowledge that agents can use to differentiate the secure reactive operations from those forbidden. Capturing security needs, the specification of policies provides agents with the information on how to behave in the correct way. The structure of these policies, therefore, must be designed with many facets taken into account.

Sometimes more than one agent in a chain of resource access processes need to comply with a set of shared policies against a specific request. For example, an agent being in an interaction may directly request system resources via an immediate partner, but it could also do so through successive intermediate system agents. The system must pass what is needed only to the authorised entities and the whole control flow must be considered before the resource access is eventually granted. Moreover, security policies should also be able to tell agents to change their reaction pat-

terns in response to requests from different social actors. Presumably, different levels of access rights to system data/services have to be and have been differentiated. Then it can be defined in policies that one agent may perform one action upon a particular service and another agent perform another, no less or more permission than that being available to them. A typical example is that retrieving or updating data may be permitted to different parties. Consequently, variation of capability roles may be played by certain system agents when resource requesters have associated with them various permission roles. Finally, agents usually perform under various contexts. The granting or refusing of access rights might be subject to the ever changing runtime context. A complete security meta-model therefore needs to be in place, with many such aspects considered.

Recognising the widely accepted RBAC and also its weakness identified in Section 2, we extend the model with an aim of meeting the unique characteristics of MAS and integrating within our agent-oriented model-driven architecture.

In our security meta-model shown in Fig. 6, each item of access permissions depends upon a set of access properties, being subject/role, operation, resource, and context. Associated with this meta-model is a security policy rule scheme that uses those components. A policy typically says a subject is granted access permission to a resource with an operation in a context. The following discussion explains the major elements that make up the security meta-model as well as its features as illustrated in the figure and in line with that, the procedure to be carried out in security aspect modelling.

#### 3.3.1. Resources under protection

In the MAS context, interacting with agents which are service invokers is regarded as a type of resource (system service) in addition to that of data and software.

Generally, a user/client agent requests a system resource from a resource provider agent through some intermediate agents as in Fig. 7. In the interaction process, one may enforce security policies just after the client agent delivers its request (in policy rule set 1) or before the resource is to be approved for use (in policy rule set 3). However, all intermediate agents provide services in one form or another and these could be targets requiring protection.

Hence, in theory, security constraints should be imposed in each agent interaction (policy rule set 1–3) rather than in a single place as many other approaches do. At least, a mechanism must be

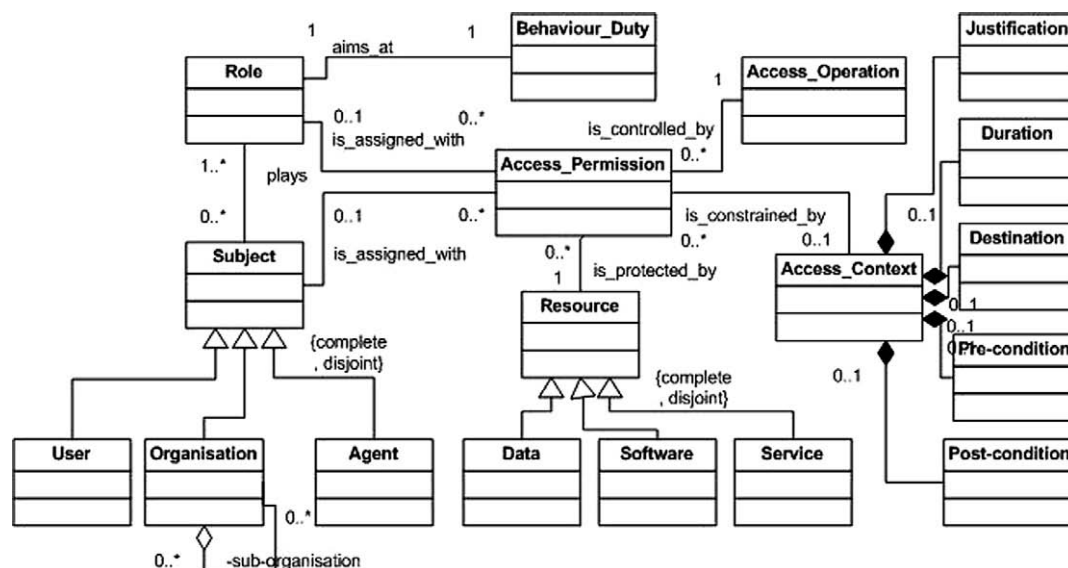


Fig. 6. The security meta-model and security-PR scheme.



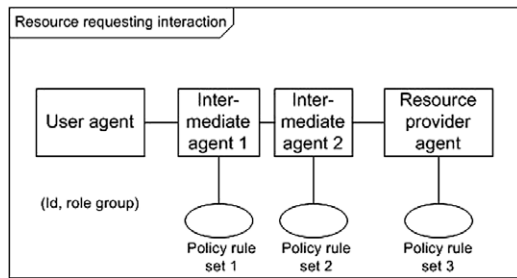


Fig. 7. A sample agent interaction model.

in place to ensure that practically, each intermediate agent behaves in such a way that reflects the policies set upon the resource access model, applicable to the request currently being sent. Suppose the above interaction is used by a house hunting user agent. It uses an estate agent finder agent (intermediate agent 1) and an estate agent (intermediate agent 2) to look for information of properties that match its preferences. An agent-finder service and a property preference matching service have been provided before the final property information is returned. Different levels of services might be provided in the process according to the user's service subscription and credit information. The permission of the contact of an estate agent does not necessarily mean it will provide all its information.

Nevertheless, in practice, it is desirable to set default access level to a group of system service providers and more restrictive constraints imposed only to certain operations upon them, as well as to critical service providers. For example, an agent offering a yellow page service could be accessed for enquiry purposes uniquely in a system by all agents but only authorised agents can update the yellow page. Likewise, all services that need to be protected and all critical operations upon services should be analysed and documented in advance, so that appropriate policies can be set for them. This corresponds to the first step we recommend in applying the modelling paradigm: analysis of services to be protected and those directly associated with sensitive data/software.

### 3.3.2. Subjects that require access to resources

Having all resources to be protected in mind, the next step is to figure out who are the potential subjects that would require access to them. Typically, they are agents running on behalf of human beings in MAS.

Actually, we can distinguish user agent and system agent as two potential types of subjects. The former needs permissions to access resources and the latter provides as well as requests services (possibly as a result of request from other agents) in MAS. System agents have tightly coupled responsibilities decided by designers. Human users have loosely coupled permissions decided by system policies. Presumably system agents are trustworthy since they are purposely designed for the system. User agents, however, can by no means be controlled in their behaviour in the system and so need to be restricted in their actions. Therefore, the analysis of interaction among user agents and those system agents which have access to critical system services and data should follow on from the previous stage.

### 3.3.3. Permission assignment on the basis of individual, role, and organisation

Having both subjects and resource objects identified. The next step is permission assignment among them.

In the same house hunting example, a potential buyer who logs on to the system will be associated with an agent running on behalf of him/her with an ID and roles. Permissions are gained by

agents through those directly associated (via the subject ID), roles they are assigned to (via subject–role relationships), or organisations they belong to (via buyer subscription group, or other organisation/membership information, etc.). The management of role definitions, user–role assignment, as well as groupings provides the maximum flexibility for access control. On an individual basis, one could have more or less rights to particular data than another of the same role. The same happens to two individuals of the same role, but from different organisations. A role hierarchy may also be defined so that some individuals from higher levels have more access rights than others from lower levels.

This scheme extends the classic RBAC96 model [19]. The advance made since then in the community has been taken into account. For instance, organisational roles of the Enterprise RBAC [36] model can now be similarly structured. Permissions are assignable to individuals as well as organisations. It might be necessary, for example, to define that certain roles can access all instances of a particular type of resource. More likely, individual entities of a resource type are specified to be accessible by individual subjects. Permissions can be assigned upon a set (or type) of resources or for a group of subjects with exceptions. This can be configured by a positive permission policy for the whole collection and a negative permission for individual exceptions.

Moreover, operational actions to services are part of the permissions. For example, for an agent holding a directory of items, one may or may not have permissions to do the following actions: enquiring, adding, updating, and removing. This is much more flexible than the limited assignment of reading/writing permission to files in some systems.

### 3.3.4. Context of access

A major advantage of the use of agents is that an agent may behave differently in different conditions, that is its environmental context. Having defined restrictions of subjects performing operations upon resources, the access context can now be taken into account, RBAC being further extended and the context concept providing additional flexibility in the scheme.

Access context may include descriptive justification of the access operation, when/where the requested data goes or the service reaches its destination of original request, the duration of the use of the data/service, the pre-condition and post-condition of the access operation. When agents play roles in interaction, the context under which they perform varies and so agents behave differently as a result of evaluating changing instance values populated at run-time. For example, an ordinary user may have access over certain regular services but in addition to that he/she may gain a premium service. This could be obtained by supplying a reasonable reason (the justification context) of the access, possibly also a commitment as a result of enjoying the service (the post-condition context), the request being eventually authorised and the access being set within a limited time period (the duration context). Context can also be used to enable access normally not seen through rights delegation. For example, two organisations can reach certain agreements in sharing their services within. One of them may delegate the use of its private services to the other organisation. Alternatively, it may delegate the access of its specific data to some particular users from the other organisation for their own use.

## 3.4. Towards a security-aware agent-oriented MDA

Both agent duties and agent rights being modelled as reaction rules and security policy rules, their integration into a single framework is essential to a security-enabled model-driven architecture. This can be achieved by using a unified role interaction model incorporating the concept in AAM paradigm and that in its supplemented security model. The principle here is.

A role plays its **functional duties** if and only if its **social rights** allow it to do so.

**Functional duty role** is dynamically bound with **social right role** before any agent plays the integrated role.

The approach deliberately separates from the beginning, the modelling of duties and rights, or functional capabilities and social permissions, in other words, what needs to be achieved in systems and how the achievement of such can be secured during that process, as RR and security-PR. Later it uses dynamic binding to put them together as AAM already did for RRs and ordinary PRs in interaction modelling.

Roles of social rights being explicitly modelled in security-PRs for MAS protects the access control to services. At the same time they must be checked against before an agent uses the services to play its role of functional duties, as RRs tell them to do so. This access control scheme fits in harmony with the existing AAM paradigm. Agents use system services to interact with each other via role playing, and as such services are in a layer below agents and invoked by them on demand (Fig. 5). At the same time, agents are a type of subject that has access to services as well as other protective system resources as permitted by associated roles (Fig. 6). So the interaction of agents via role playing as well as role restricting ensures the use of critical system services always complies with rule regulations. The overall model becomes complete in the sense that the complementary role nature of functional duties and social rights are both taken into account seamlessly. Policies can be defined separately from the main functionalities and applied by these agents to check the requests before services under protection are provided.

The dynamic binding of RRs and PRs towards a complete role-based model adds maintainability and adaptivity to systems. The original distinction of rule types in AAM is motivated by the rationale that requirements, with regard to locally affective agent behaviour (as modelled in RRs) and the requirements with regard to globally affective agent behaviour (as in PRs), can be managed separately. The separate reconfiguration of them allows one to specify individual agent reactive behaviour per RR, or all agents' policy conformation behaviour per PR, both types of rules being automatically deployed once the re-configuration is complete. Considering the special PR type of security policy rules, social rights mapped from non-functional security requirements can be specified separately from functional capabilities mapped from ordinary functional requirements. This eases the model specification process for human experts. The interaction model, capturing all functional roles, can be defined in the first place and then, on top of that, the policy rule model constrains the execution of them to a group of authorised agents. This clearly splits the modelling tasks, but the association of them via an integrated RR and PR scheme, including use of the role concept to pull together rights and duties, avoids any potential perturbation. Only when an agent comes to perform a role at runtime, aiming at contributing its functional capabilities as described in RRs, is the role of this agent assigned and the social rights checked against all applicable security policies in the PRs.

## 4. Case study

### 4.1. The requirements specification of a British railway management system

To demonstrate the efficacy of our approach, we have investigated how it might be applied to an actual system, the British railway management system called Railtrack. The system is mainly responsible for the running of a railway on a daily basis, monitor-

ing 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 and safety critical system, the documented specification has more than 250 pages and contains a large number of function descriptions in a unified form. An extract from the original specification is presented in Fig. 8 and several standardised function descriptions are shown in Table 2.

A UML activity diagram, showing the activities and flow of control for the case study scenario, is given in Fig. 9. It demonstrates the handling of faults, the subsequent imposing of restrictions, and rescheduling of train services which together compose the process of fault management. Participant actors collaborate and perform activities sequentially and conditionally, forming a control flow of the system. Activity Diagrams provide a task-centric view of the behaviour of objects. When object-oriented systems are being implemented, activities are statically assigned to objects and the message passing patterns between objects in the systems are fixed. UML activity diagrams are not inherently object oriented and, due to their flowchart heritage, mapping to object-oriented concepts is difficult [37]. The figure also shows some constraints imposed upon *IMI-HandleFault* in the form of object constraint language, asserting business rules on top of UML models [38]. The pre-conditions, post-conditions, and invariants are assertions that can be expressed in OCL and abstracted in order to facilitate implementation. However, OCL elements are attached to other modelling elements, separated from the main control flow rather than integral components. This is due to the lack of support for specifying constraints in activity diagrams or use cases. Sometimes, it is even hard to use OCL to describe the constraints that need to be imposed on the design models. For instance, not all Contractors can perform the task of *FixFault* upon a given track with the associated access power to it. Security constraints must be defined for proper resource management. Therefore, current OO design methods have some limitations in modelling a system in an adaptive architecture that completely captures the structure and behaviour of the system and later drive its running directly. Adaptation to the business-related functions such as *HandleFault*, security-related constraints such as Contractor's access to the rail tracks, and the overall integrated architecture will usually entail major maintenance burden and difficulty in the life-cycle of the system. Later, we will compare our development approach against conventional methods.

Before the presentation of the approach, it may be worthwhile noting in the case study that external actors may provide services outside the boundary of the Railtrack specification. Contractors, for example, offer track maintenance services. They may be requested to fix the faulty tracks when the system detects them, by passing messages. The maintenance of such services is not up to Railtrack but according to contracts. As far as the system is aware of the existence of such services and their interfaces, they can be made use of even in a distributed environment.

A primary goal of the system is delivering train services safely. While one may understand this as a system goal of keeping train services running and a security goal of keeping the running services safely, we shall by no means isolate them although we can model them separately but later integrate them together. Actually, that is where functional duty roles and social right roles come into play in union. In the next sections we will start from requirements analysis, and build models based on them towards automatic agent execution. Functional requirements and non-functional security-related requirements are both considered and integrated in one unified interaction and policy model.

### 4.2. Building the CIM: requirements analysis

Requirements analysis will be carried out in this section for building the computation-independent model.

The specification (“Production Function”) comprises three main areas: **Train Running and Performance**, **Infrastructure Management and Performance**, and **Common Communications**, each of which is sub-divided into *Business*, *Incident*, and *Execution* domains. These areas or domains are closely linked. For example, the police phoning in a report of a vehicle on the line requires drivers to be notified (Common Communications), an emergency isolation and possession to be taken (Infrastructure Management) and trains to be stopped or re-routed (Train Running).

Train Running Business (**TRB**) domain supports the principal service to customers, including delivery of planned train paths and response to requests for further train paths. Relating to the domain, **Train Operators** run train journeys on the network. They are normally freight or passenger train operating companies. Each train journey is first supplied in the form of a plan, either as part of the working timetable (Requirement: *TRB-AcceptTimetable*), or as a result of a late request from a customer (Requirement: *TRB-AcceptLateAddition*). In some cases, **Drivers** can gain permission to make a journey from depot and a plan must be drawn up for the train and a train journey defined for it (Requirement: *TRB-AcceptLateAddition*). Later the planned train journeys could be diverted to avoid engineering works or as a result of response to incidents in which cases there will be a timetable update (Requirement: *TRB-AcceptTimetableChange*). These functions handle changes to the train plan before the affected train services are scheduled to start.

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/incidents will cause train service re-planning. With respect to the management of infrastructure and faults as well as the rescheduling of affected train services, involved domains 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 when incidents occur. External entities include: **Train Operators**, who initiate train running requests, in this context of fault management, have to be consulted when dealing with perturbations; and **Contractors**, who carry out maintenance.

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*). But an asset fault is not the only reason that could cause track restrictions. An external person on track may need to be protected by a track restriction, for example, an emergency isolation. Such a person may be either an **Unauthorised Person** (or even an **Animal**) reported to be in some location (Requirement: *IME-ImposeSuddenRestrictions*), or a person requiring access to the track, for example, **Contractor** staff (Requirement: *IME-ImposeRestrictions*). All these situations will involve changes to forecast train journeys (Requirement: *TRI-RespondToIncident*). Affected train journeys are amended for re-scheduled services to the **Train Operator**. As time passes or work progresses, further information may be received about the fault (Requirement: *IMI-UpdateFaultInformation*). This may result in changes to the priority of the fault or imposition or removal of track restrictions. A special case of this is the final fixing of the fault, when the restrictions will be removed.

Fig. 8. An extract of the functional specification of the Railtrack case study.

#### 4.2.1. Requirements analysis: functional requirements

In the case study, requirements are organised into domains, which in turn consist of domain functions. Apart from the functions that are dedicated to regular train services, others span over several business domains and coordinate in order to manage faults/incidents and make rectification wherever necessary. This is critical to the safe running of train services. Planned train services that will later use the faulty assets may have to be rescheduled, and once assets recover from faults, train services should come back to normal. The system must ensure safe train services, as well as their normal running in an optimal timeline. Therefore, appropriate access control to system resources is necessary. On one hand, the system needs to timely respond to faults and reschedule train services to prevent disasters. On the other hand, it needs to efficiently protect critical system resources such as train journeys and rail tracks so that train services can be as according to normal schedule as possible. Thus, access to those resources must be authorised. In exceptional situations, unauthorised access is allowed but must be restricted, people or animals being prevented from accidental disasters.

External actors, being people, organisations, objects, or other systems may interact with the Railtrack system, in the processes of which authorised resource access must be guaranteed. Often

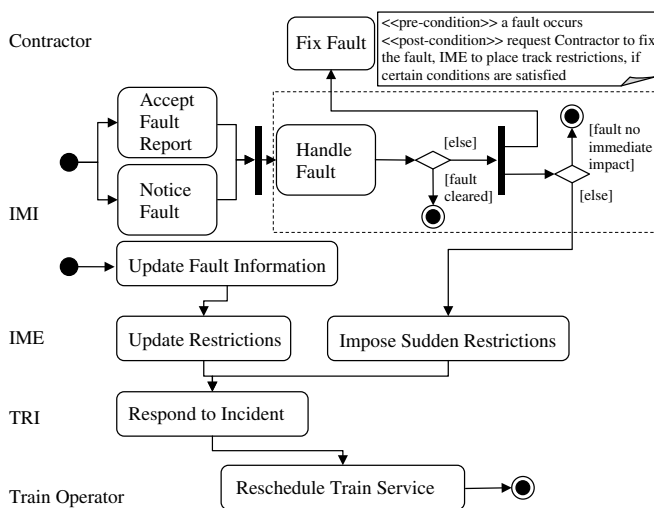
an actor represents a number of different things all of which may be considered the same because they play the same role in a particular process but when the same actor occurs in different processes it may play different roles. For example, a driver appears as an actor in their own right in the train running process because they have specific roles in train running. They may also be people who report incidents in which they assume a different role.

Inside the system (specification), IMI, IME, TRI, TRB are some business domains presented in the case study. Delegated to respective agents, these can represent internal actors that are responsible for the function of their respective business domains, who have the knowledge concerned with those domains and responsible for realising domain functions. They respond to external actors as well as interact with each other internally. Table 3 summarises the roles of external actors and business domains represented by their corresponding agents, through the interaction of which system functions are fulfilled. Domains roles are collective functions such as *IMI-HandleFault* given in Table 2.

Functional requirements specification is thus organised, domains being represented by agents that act on behalf of them, and domain functions collectively forming duty roles which agents play. Before non-functional requirements which are implicitly

**Table 2**  
Sample functional descriptions from the specification

Domain	TRB	IMI	IME
Identifier	AcceptLateAddition	HandleFault	ImposeSuddenRestrictions
Description	To handle a late request for a train journey.	To maintain information about faults so that they can be fixed in a way which minimises the overall impact on the business	To protect people on the track by protecting designated locations
Cause	Receipt of a request for a train journey directly from a <b>Train Operator</b> or from the <b>Driver</b> of a train entering the Production Function's area. The request is provided in the form of a combination of relevant train details, locations and desired timings	A fault becomes known to the Production Function, either from people reporting information about a fault ( <b>IMI-AcceptFaultReport</b> ), or directly from the infrastructure asset, via infrastructure monitoring equipment or from failure to operate when commanded ( <b>IMI-NoticeFault</b> )	Immediate authorised access to track is required ( <b>IMI-HandleFault</b> ), or there are unauthorised people on the track
Assumption	The crew is competent for the route requested	–	Accurate information about the locations which need protecting is available
Information used	Relevant locations	Information about infrastructure assets and their contracts Information about train journeys to assess the impact of the fault	Information about locations and the associated infrastructure assets Information about train journeys
Outputs	A new train journey, to the <b>Train Operator</b> and others	Fault information to <b>Contractors</b>	Commands to protection system and electrical network Information about protections put in place to people requiring access to track
Required effect	A new train journey is created from the request, and validated ( <b>TRB-ValidateTrainPlan</b> ) If the train journey is acceptable then it is distributed to all interested parties; otherwise the request is rejected or renegotiated Having been accepted, the new train journey is known to the Production Function	The fault is recorded  Unless the fault has already been cleared, the appropriate <b>Contractor</b> is identified and agreement is reached about a priority for fixing the fault If the fault is associated with an existing incident, then that is recorded; otherwise, if it has some impact then a new incident is established with the fault as its cause If necessary, track restrictions are put in place ( <b>IME-ImposeSuddenRestrictions</b> ). If so there is an impact on the train service handled by <b>TRI-RespondToIncident</b>	Track restrictions are created  The effect on train journeys is handled ( <b>TRI-RespondToIncident</b> )
Comment	Common examples are light engines and empty stock movements which are not planned  Sometimes trains appear without warning, e.g., from a depot and the <b>Driver</b> requests permission to make a journey. This needs to be treated as if it is a request from the <b>Train Operator</b>	–	Ground level production staff may use a key to manually replace signals to danger



**Fig. 9.** An activity diagram describing the case study.

**Table 3**  
Roles of actors and domains

Actor/Domain	Role
IMI	Detect and handle asset faults
IME	Place track restrictions
TRI	Handle the impact of incidents on train journeys by the amendment of those affected
TRB	Deliver planned train paths and respond to requests for further train paths or amended ones due to incidents
Contractor	Fix asset faults
Train operator	(Re-) schedule train services

top of which policy rules can then be modelled for dictating the appropriateness of role execution according to role permission assignment. To this end, a typical domain function *HandleFault* that IMI agent must fulfil is now presented in narrative as in Fig. 10, prior to our overall model building process.

Restructuring domain function requirements (already described in Fig. 9 and documented in Table 2) in this form supports later modelling using our unified reaction rule scheme. This is because generically, a domain function captures a reaction behaviour where an event triggers an agent to perform an action using a decision making process. Thus, the specific function will become a

embedded in the specification are represented by social right roles, the interaction model of agent duty roles must be structured, on



```

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

```

Fig. 10. The restructured domain function *IMI-HandleFault*.

reaction rule “*HandleFault*” for agent “IMI”. The decision making process the RR uses for fault handling when one is reported is shown in Fig. 11. This corresponds to the scheme shown in Fig. 5. A branch of condition2 → condition2.2 will be selected to request the fixing of fault (fault not cleared) and place track restrictions (fault has immediate impact) if corresponding conditions are met.

Straightforwardly, agent IMI processes its RR *IMI-HandleFault* in a manner shown in Fig. 12. A “Fault” and an “Asset” component will be selected to facilitate the agent to function in the process.

When many such RRs are structured from the original specification capturing agent duty roles, their interaction model can be documented capturing the blueprint of the overall system model. Section 4.3.1 will describe such a model. Before that, the non-functional security requirements need to be analysed with an aim of integration into this interaction model.

#### 4.2.2. Requirements analysis: non-functional security requirements

IMI agent is a system agent and itself performs no direct operation upon critical system resources. However, it could act under the request of people playing different roles, e.g., when a fault report is received, and then it needs to interact with its partner IME, to place track restrictions to rail tracks. This could have a major impact on train services the system is aimed at delivering. Such access to the train tracks can only be granted if the report about the faulty track is validated. For example, drivers working in the railway environment can provide first-hand information about the changing rail track condition and can therefore be trustworthy in reporting emergencies. They are, however, could be constrained

by making additional train journeys without authorisation. A mechanism must be in place for judging which roles should have access (and the level of access) to critical system resources when they make requests.

In terms of the direct impact IME has upon the critical system resource of rail tracks via its operations, the system needs to assign different permissions to two types of social roles, Contractor and Unauthorised People, both of which access tracks via IME. IME plays the role of placing track restrictions functionally without differentiation of resource requesters. However, the role it plays has two variations via binding to the social role of the different requesters who have made request to this function. In other words, Contractor and Unauthorised People are the two social roles that request the access to tracks via IME and because they have different levels of access to system resources, IME should assume different social roles. It assumes its role of *ImposeRestrictions* on receipt of a request from the former, giving Contractor permissions to do appropriate repair operations upon track and, assume its role of *ImposeSuddenRestrictions* on receipt of a request from the latter, giving Unauthorised People access to track within limited time period. In two occasions, IME actually binds its same role of functional duty with one of the two different versions of the social roles determined by the upcoming requesters. Consequently, the role of functional duty is uniformly conformed with and the role of social rights is also respected. Using this scheme, the resource access via a number of intermediate system agents as discussed in Section 3.3 is well controlled.

The above procedure protects system resources from the view of a successive interaction processes. More comprehensively but not necessarily more complexly, requirements analysis towards a secure system development follows the simple procedure as in accordance with the security model presented in Section 3.3.

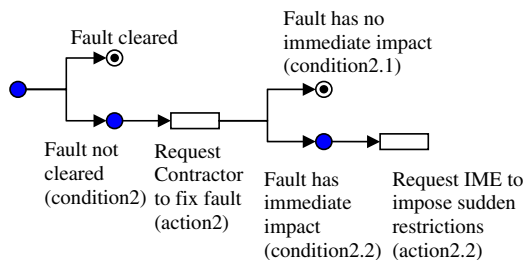


Fig. 11. The decision making process tree structure of *IMI-HandleFault*.

- Identifying entities involved in critical service delivery and data access as required by the specification.
- Mapping them to various classes of entities using the security meta-model describing the relationships among them, reflecting the actual concrete constraints.
- Formulating security policy rules based on roles for later mapping and binding.
- Associating security policy rules with functional interaction model, which already centred on agent role playing behaviour.

- Step1: Receive fault report message from “AcceptFaultReport” or “NoticeFault” from the same agent.
- Step2: Construct a “Fault” and an “Asset” object using the information contained in the message.
- Step3&4 {condition, action} couplet1: If the created “Fault” object is evaluated by the “cleared()” method as FALSE (Condition2), Then send a message with the created “Fault” to “FixFault” owned by Contractor agent (Action2), and
- Step3&4 {condition, action} couplet2: If the created “Fault” object is evaluated by the “immeImpact()” method as TRUE (Condition2.2), Then send a message with the created “Asset” to “ImposeSuddenRestrictions” owned by IME agent (Action2.2).
- Step5: Add the belief that a fault occurs at this moment with a potential incident related with it.

Fig. 12. Brief steps of RR *IMI-HandleFault* processing by agent IMI.

- The access control model eventually becoming part of the now restrictive functional interaction model with security constraints set upon.

Following the simple procedure, we can identify and formulate, using the case study description, the potential subject accessing resource patterns. (1) Resources to be secured include: train journey timetable/plans, infrastructure assets/tracks, etc. (2) Subjects that require access to resources include: IME to tracks (via imposing restrictions), TRB to train journeys (via amending timetable), etc. They have direct access to resources and also indirectly on behalf of others who request such access. For example, IME may grant Contractor staff access to tracks for repair purposes. These staff can be assigned with the role “Contractor”, distinct from others, e.g., role “Unauthorised People/Animal” who may get in tracks by accident and then be granted temporary access in the time period of which their lives are protected before they leave the tracks. (3) Access operations are in accord with each individual type of resources. In the above case, Contractors may have two kinds of operations to tracks: access and repair. (4) Access context for each particular access may vary. If a contract has been signed between a Contractor and the train operator company at present, the Contractor can access and repair the tracks it is responsible for maintenance of but may not have access to the rest of the rail assets.

Next, security policy rules can be set up using the identified entities and their interrelationships, as in accordance with the actual business needs, on the basis of the security meta-model and security-PR scheme shown in Fig. 7. One example rule already mentioned could be: Unauthorised People/Animal gains access to tracks within a limited time period. Another one may be: Contractor gains access as well as repairs tracks they are responsible for under certain contracts. Both of the two roles, Contractor and Unauthorised People/Animal would request the access to tracks via IME and they are constrained respectively by these two security-PRs. IME will at runtime bind its uniform duty role of imposing track restrictions to the proper social right role with which it runs in the name of. In this way, the setting and resetting of policies according to our changing needs results in the assignment of changing permissions to those actually in need of different levels of access from time to time. This process is separate from the configuration of RR. When resource requesters of affected social right roles request the access, the intact functional duty role under request is performed identically but immediately the subjects retain the appropriate permissions of access that we just set.

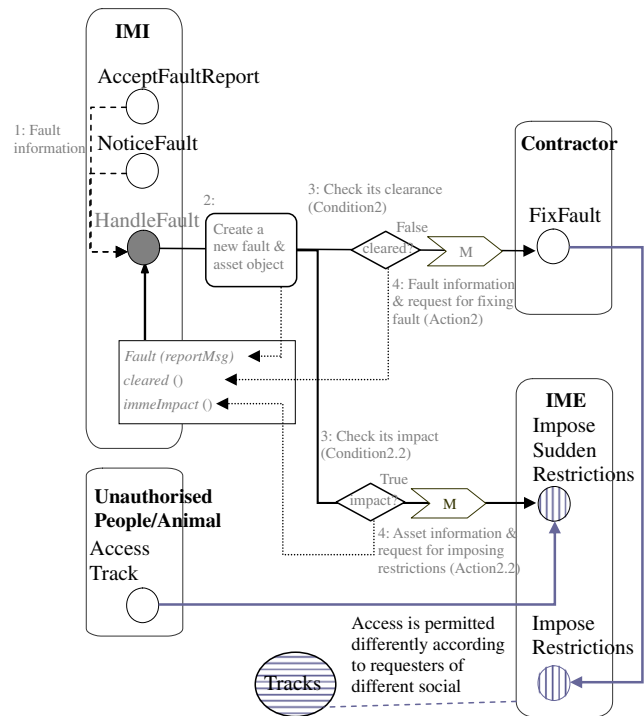
#### 4.3. Building the PIM

This section will demonstrate the construction of the platform-independent model.

##### 4.3.1. Design model: functional duty role model

Fig. 13 describes part of the interaction towards the goal of handling faults. Involved in it is the required agents and their associated rules, collectively they make decisions via service facility and perform reactive behaviour via message passing. Together these model elements shape the control structure of the interaction. The narrative description of the rule *IMI-HandleFault* obtained in requirements analysis phase (Section 4.2.1) is used, among its other peer elements, for building a fully connected interaction model.

The binding of RR with security-PR together configures role playing variations when system resource access is requested by various social right roles. When Contractor and Unauthorised People/Animal require access to rail tracks via IME, IME plays the identical RR “*ImposeTrackRestrictions*” functionally but grants respective access rights to them. This can be regarded as two roles



**Fig. 13.** An interaction model showing *IMI-HandleFault* functional duty role's compositional elements (lighter print) and IME varies its social right roles while requested through different resource access routes (blue stripes). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

of *ImposeSuddenRestrictions* and *ImposeRestrictions* produced as the result of binding are played effectively (with the same function but different access permissions). The specification of the RR “*ImposeTrackRestrictions*” is given in Fig. 14 and annotated XML structure in Fig. 15 for agent comprehension and execution. It is assumed that “*Restriction*” is a service publicly available to all agents.

A guideline for transforming functional requirement tables (as in Table 2) to RR structures and then XML specifications is provided in [27]. In it a process that each agent reacts to the receipt of a message by executing the given rule structure is also presented. Those would later be extended so that when IME uses its one and only RR, the two externally linked security PRs are dynamically played, forming two combinational different duty and right roles. The overall interaction model is shown in Fig. 16, low level model elements being ignored.

##### 4.3.2. Design model: social right role model

Some security policy rules may simply say access requests to particular resources from certain roles will be accepted or rejected, implying the successful completion of interaction model is subject to the role of the subject who initiates the model. This makes perfect sense since sometimes we simply want to restrict the execution of our model to those who need them indeed and safety is achieved in this manner.

In the Railtrack scenario, drivers are capable of updating train journeys functionally by making late requests, e.g., *AcceptLateAddition* is a role played by TRB dedicated to respond to such requests (shown in Fig. 16). Implicitly, this role performs an add action to the resource type of train journey, which is a critical system resource. The social rights of drivers, however, may constrain their functions as such spontaneously. This is supported by the rationale that, unplanned journeys may conflict with existing ones sharing the same tracks and without the complete knowledge of these,

<b>Reaction Rule: ImposeTrackRestrictions</b>
<b>Owner Agent:</b> IME
<b>Use Context:</b> Interaction model: Request track restrictions
<b>Role Description:</b> Impose restrictions to tracks when incidents occur, faults are detected, or people get on track by incident
<b>Role Playing:</b> Triggering Event: Respond to: Contractor or Unauthorised People/Animal Incoming message: Track restriction request (request: req) Processing: Component/Service use: Restriction Invocation call: res = createNewRestriction (req) Return result: Restriction res Further Action: Request to: TRI Outgoing message: Request the handling of the effect to train journeys (Restriction: res)

Fig. 14. Reaction rule “ImposeTrackRestrictions” specification.

```

<reaction>
  <name>ImposeTrackRestrictions</name>
  <interaction-model>Request_track_restrictions</interaction-model>
  <owner-agent>IME</owner-agent>
  <global-variable>
    <var>
      <name>req</name>
      <type>Request</type>
    </var>
    <var>
      <name>res</name>
      <type>Restriction</type>
    </var>
  </global-variable>
  <event>
    <message>
      <from>
        <from>Contractor.FixFault</from>
        <from>Unauthorised_People.AccessTrack</from>
      </from>
      <content>
        <request>
          <location>London</location>
          <asset>
            <id>10015</id>
            <type>rail</type>
            <contractor>contractor_10</contractor>
            .....
          </asset>
        </request>
      </content>
    </message>
  </event>
  <processing>
    req = new Request (requestMsg)
    res = Restriction.createNewRestriction (req)
  </processing>
  <action>
    <message>
      <to>TRI.RespondToIncident</to>
      <content>
        <restriction>
          .....
        </restriction>
      </content>
    </message>
  </action>
  <priority>5</priority>
</reaction>

```

Fig. 15. XML annotations of reaction rule “ImposeTrackRestrictions”.

disasters may happen. Nevertheless, we are quite happy if freight drivers just want to make empty stock movement from a depot without any effect to existing train journeys, such unplanned operations being regarded as safe. In other situations, such requests should not be authorised in order to protect both rail tracks and

planned train journeys. In Fig. 17 Policy rule 001 defines a constraint that only allows drivers to make empty stock movement from a depot and policy rule 002 further constrains this with additional conditions that must be satisfied if drivers want to make additional train journeys. It should be noted that when drivers do something else in the system which is not associated with the driver role, that action must be constrained by other associated policy rules. For example, all drivers can report faults on tracks as a role playable by all people.

Sometimes returning a result of acceptance or rejection is insufficient, and levels of access rights must be differentiated. Contractors and Unauthorised People/Animals both can access tracks but have different levels of access rights, as already shown in previous sections. Policy rule 003 and 004 formalise the role access model in Fig. 18.

Fig. 13 illustrates agent IME using a RR “ImposeTrackRestrictions” to deal with different potential resource users, as also shown in Fig. 14. That RR maintains a unified behaviour model. When Contractor and Unauthorised People/Animal come into play, security PRs are externally evaluated and the relevant individual bound with the RR for execution. The result of executing the combination of the RR and separate security PRs makes a difference to the original functional interaction model. Implicitly, *ImposeSuddenRestrictions* or *ImposeRestrictions* roles are assumed in two situations. A role of *ImposeRestrictions* is bound in one occasion and Contractor staff can access tracks with additional repair operations. The other role of *ImposeSuddenRestrictions* is bound in the other occasion and Unauthorised People have access to tracks but within a limited period of time. Role of functional duty is thus discharged uniquely by IME but the appropriate permissions are granted depending upon the role of social rights of requesters IME runs in the name of.

When different security PRs must be applied to different social right roles, it is also convenient to configure a RR to internally perform different actions in addition to return different permissions to resource requesters. This can be easily done by setting different access requests to pass through different decision making tree branches (so mapped to different {condition, action} pairs) and associated with those the use of different levels of services (either various versions or various internal use). Essentially, PRs are incorporated as part of the RR branches, mapping to the roles of social rights to which they are to be bound. The existing RR structure scheme allows easy accommodation of conditions mapping to various social right roles and actions mapping to various service operations and access permissions. The configuration of different versions of services for different resource requesters is also possi-





**Goal 003:** Allowing Contractors access to tracks to e.g. repair faulty tracks under contracts.

**Security Policy Rule 003:** Contractors can make authorised access to train tracks.

**Security Policy Rule Instance 004:**

*Contractor (Role) contractor\_10 (Subject) can access and repair (Access Operation) rail track track\_01 (Resource) if a contract has been signed in which it states some faulty tracks are under the maintenance of this contractor and track\_01 is among these or those associated with them (Access Context).*

**Goal 004:** Allowing Unauthorised People/Animals access to tracks but constraining such access to a minimum time period to limit its impact to ordinary train services. Further actions must be taken place to protect the involved parties as well as clear the tracks after temporary access has been assigned to the unauthorised access.

**Security Policy Rule 004:** Unauthorised People/Animal can access train tracks temporarily.

**Security Policy Rule Instance 005:**

*Unauthorised People/Animals (Role) upa\_11 (Subject) can access (Access Operation) rail track track\_01 (Resource) if they are on the track accidentally but this access is within limited time period (Access Context).*

Fig. 18. Security-PRs associated with the rail track access control.

```
<Security_Policies>
  <Policy id="p_003">
    <Affection>
      <Role>Contractor</Role>
    </Affection>
    <Permission description="Allowing Contractors to access tracks to
    e.g. repair faulty tracks under contracts" given="yes">
      <Subject id="contractor_10">
        <Role>Contractor</Role>
      </Subject>
      <Access_Operations>
        <Access_Operation>access</Access_Operation>
        <Access_Operation>repair</Access_Operation>
      </Access_Operations>
      <Access_Context>
        <Justification>
          a contract contract_01 has been signed in which it states some
          faulty tracks are under the maintenance of contractor_10
          and track_01 is among these or those associated with them.
        </Justification>
      </Access_Context>
      <Resource id="track_01">
        <Type>Rail Track</Type>
        <Location></Location>
      </Resource>
    </Permission>
  </Policy>
  <Policy id="p_004">
    <Affection>
      <Role>Unauthorised People/Animals</Role>
    </Affection>
    <Permission description="Allowing Unauthorised People/Animals to
    access tracks but constraining such access to a minimum time period
    to limit its impact to ordinary train services.
    Further actions must be taken place to protect the involved parties
    as well as clear the tracks after temporary access has been assigned
    to the unauthorised access." given="no">
      <Subject id="upa_11">
        <Role>Unauthorised People/Animals</Role>
      </Subject>
      <Access_Operations>
        <Access_Operation>access</Access_Operation>
      </Access_Operations>
      <Access_Context>
        <Justification>they are on the track accidentally</Justification>
        <Duration>
          <Start_Time>ontrack_detection_time</Start_Time>
          <End_Time>track_clearance_time</End_Time>
        </Duration>
      </Access_Context>
      <Resource id="track_01">
        <Type>Rail Track</Type>
      </Resource>
    </Permission>
  </Policy>
</Security_Policies>
```

Fig. 19. Security-PRs annotated in XML.

principles. In this way, MDA is applied when the PIM is obtained upon the completion of RR and security-PR modelling in integrated interaction models.

Fig. 20 shows the platform-specific model being transformed for the JADE platform. The concrete behaviour classes in association with JADE agents are in one way based upon the functional RRs and in another way constrained by security-PRs which determine their behavioural permissions. The constructs required by JADE as well as other supplementary classes are illustrated in the figure. Specifically, (1) addRole and addBehaviour methods put the base behaviour into the JADE agent envelope; (2) a base behaviour is associated with the RR components of event and action according to the RR scheme; (3) the actual access permission of the request for resources as captured in the event will proceed if the security-PR set returns a positive outcome; (4) the consultation result of the permission class of the behaviour module is decided by the applicable security-PRs to the resource requester; (5) the dynamic security concern determined by PRs is taken into account in the actual access behaviour raised by runtime events in the behaviour class.

The PSM can be further transformed to code, executable using the JADE platform, core classes including event and action message, policy rule Set, as well as permission built in PSM being used. The MDA practice of mapping/transforming from abstract models to concrete models requires the former models to contain sufficient knowledge details about the system being modelled. Accompanied with XML specifications, the models we build contain sufficient semantic details of system structure, behaviour, as well

as constraints. Both functions and security constraints captured in the formalised models can facilitate agents to interpret from these constructs their behaviour. Agents can uniformly interpret from the RRs' XML specification (as shown in Fig. 15) of their duties, as well as from the security-PRs' XML specifications (as shown in Fig. 19) of their rights. According to the RR scheme, an "event" element maps to a conditional check of the incoming message source against the one defined in the current rule specification. An "action" maps to a message passing behaviour and so on. According to the PR scheme, an access "role" element, the "resource" element under request, as well as the access "operation" element, as requested by the event will be checked against the applicable policy set in a loop condition. The combined behaviour is thus achieved in a structure, such as the one shown in Fig. 21, so that when a specific event occurs that matches with a RR defined to handle the event (code fragment from a to b), its action will be taken (code fragment from c onwards), if the associated security-PRs that have been defined to constrain this RR are found to be satisfactory (code fragment from b to c).

## 5. Model-driven adaptation (after deployment)

The security-enabled AAM approach achieves both adaptivity of functional duties via functional interaction model (add new collaborators, new service facilities, new roles, etc.) and adaptivity of social rights via an integrated security policy rule model (add new security PRs, etc.).

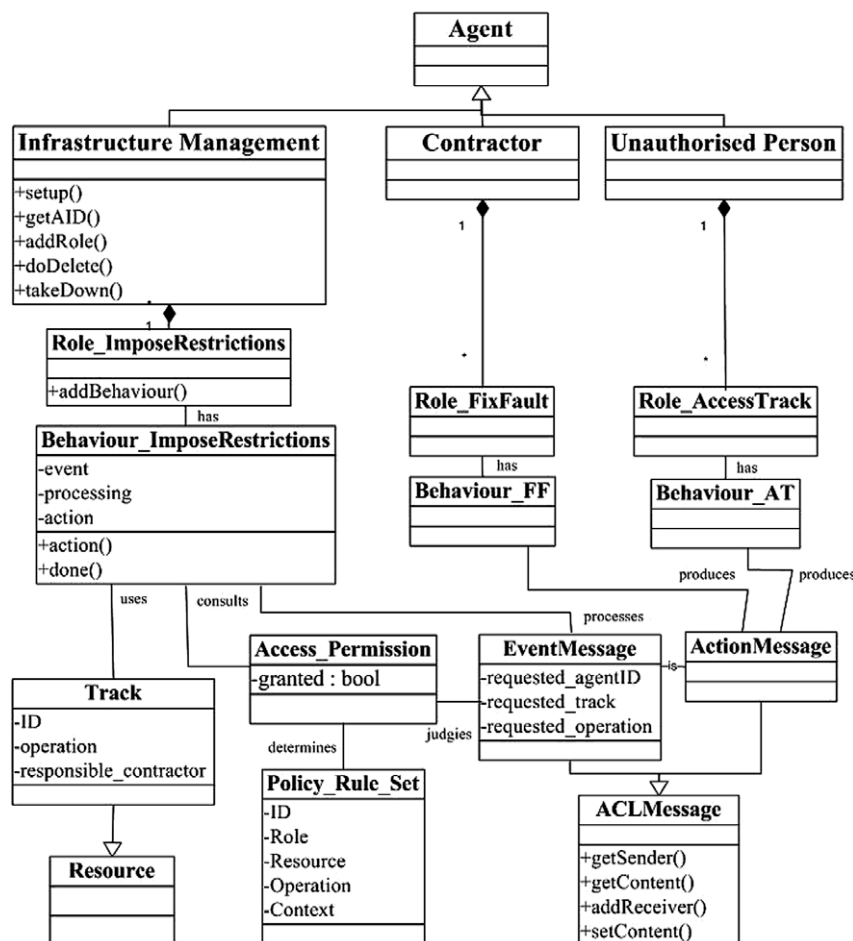


Fig. 20. PSM of the case study, composed by the RRs and security-PRs.

```

Infrastructure_Management.addBehaviour (FunctionalRole ImposeRestrictions) {
BehaviourRule behaviourRule = ImposeRestrictions.getBaseBehaviour();
Request req;
Restriction res;
Message m = thisAgent.receiveMessage ();
while (m != null)
{
    Agent fromAgent = m.getSenderAgent ();
    // a) The event message is coming from an expected source
    if (fromAgent.equals
        (behaviourRule.getEvent (). getMessage (). getFromAgent ()))
    {
        XMLSchema schemaIn =
            behaviourRule.getEvent (). getMessage (). getSchema ();
        XMLSchema schemaOut =
            behaviourRule.getAction (). getMessage (). getSchema ();
        ObjMsg requestMsg =
            m.getContentObject (). unmarshal (schemaIn);
        req = new Request (requestMsg);
        res = Restriction.createNewRestriction (req);
        /* b) All applicable policy rules permit to activate the RR */
        RequestPermission per = new RequestPermission (fromAgent.getSocialRole (),
            req.getResource(), req.getOperation ());
        PolicyRuleSet applicableRules = Policy_Rule_Set.getApplicableRuleSet
            (fromAgent.getSocialRole ());
        if (per.check (applicableRules)) {
            // c) Execute the RR action
            if (res != null)
            {
                XMLMsg res = e.marshal (schemaOut);
                Message m2 = new Message ();
                m2.setContentObject (XMLMsg);
                Agent toAgent =
                    behaviourRule.getAction (). getMessage (). getToAgent ();
                m2.addReceiverAgent (toAgent);
                thisAgent.send (m2);
            }
        }
    }
    m = thisAgent.receiveMessage ();
}
}

```

Fig. 21. Pseudo code of the case study.

### 5.1. Adaptivity of functional duties

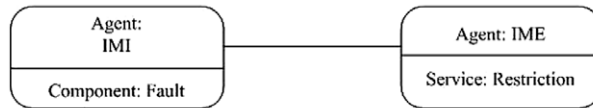
First of all, the AAM achieves adaptivity both in inter-agent interaction and intra-agent computation. At the time of running, the interaction model has its associated RR definitions distributed among all involved agents and individually available to them once an interaction model designer creates such a model. Update of the model pushes the updated rule definitions to the relevant agent sites. The formation of interaction model is dynamic and the rule execution on the fly immediately reflects the required change. For example, human experts can associate a different “Restriction” service to “ImposeTrackRestrictions” as shown in Fig. 16 diagrammatically or in Fig. 14 textually. This makes IME to transparently apply a new service reflecting new internal policies for establishing restrictions. Similarly, agent partnerships and multi-agent role playing collaborations can be configured. For instance, when a driver detects a fault, it is through interaction of *IMI.HandleFault*, *IME.ImposeTrackRestrictions*, *TRI.RespondToIncident*, and *TRB.AcceptTimetableChange* that affected train journeys are rescheduled eventually. A shortcut path of interaction can be specified for special occasions so that other parties are not required to be involved. For example, a driver may need to directly alter an affected train journey under his/her own operation and redirect the journey through an alternative path if the driver detects the original planned path is being occupied temporarily by, say, an animal intrusion incident. In this way, the system runs much more flexibly.

These can be achieved due to the fact that, agents actually come into play in interaction model only when they are triggered by event messages on the fly. Rules that are defined to deal with the matched events are dynamically retrieved from a rule repository, always up-to-date. It is from these rules that agents discover the roles they should play at runtime. A set of rules can be defined with an order of priority. Rules with lower priorities will be overridden by those with higher priorities, according to prioritised business

needs or exceptional situations. For example, two versions of the interaction model might be available while the only difference is in one of the rules associated with the models. A rule with a lower priority tells its agent to behave in the interaction in a general situation. The other rule with a higher priority tells the same agent to perform in the interaction on more specialised scenarios. Individual rule specification allows flexible and fine-grained model configuration.

More comprehensively, Fig. 22 illustrates the adaptive inter-agent collaboration dependency and intra-agent computation capability achieved through the AAM modelling approach. The upper section shows the relationship between two collaborative agents in the case study interaction model. IMI uses *HandleFault* to collaborate with IME, which responds using *ImposeTrackRestrictions*. Components and services are invoked during the function of two rules, respectively, assisting the agent function. This might cause the illusion that the relationships between agents and the agents’ use of computational entities are fixed. In fact, there is no direct link between agents, or between agents and components/services. Rather, such collaboration relationships are specified in the externalised rules, which are dynamically retrieved. In other words, it is at the time that a rule is selected an agent knows its collaborative agents and supporting components/services. The configuration of two rules can change not only the collaboration relationship between IMI and IME, but also the use of “Fault” component and “Restriction” service. Such information is completely transparent to agents and only known to them at the time of their activation by events. Thus, once an agent is requested for participation, the other agents it will interact with and components/services it will use for computation in the following phase are dynamically decided and replaceable at runtime by configuring rules, with immediate effect. In the overall model perspective, the individual rule running process progressively forms the interaction model, sometimes according to the previously computed results. Connec-

Inter-agent dependency and their internal computation models have been hard-coded traditionally



*We break such a tradition by introducing an interaction model where agent role playing, computing, and message passing involved in the interaction are all captured in structured rules, being interpreted by agents at runtime*

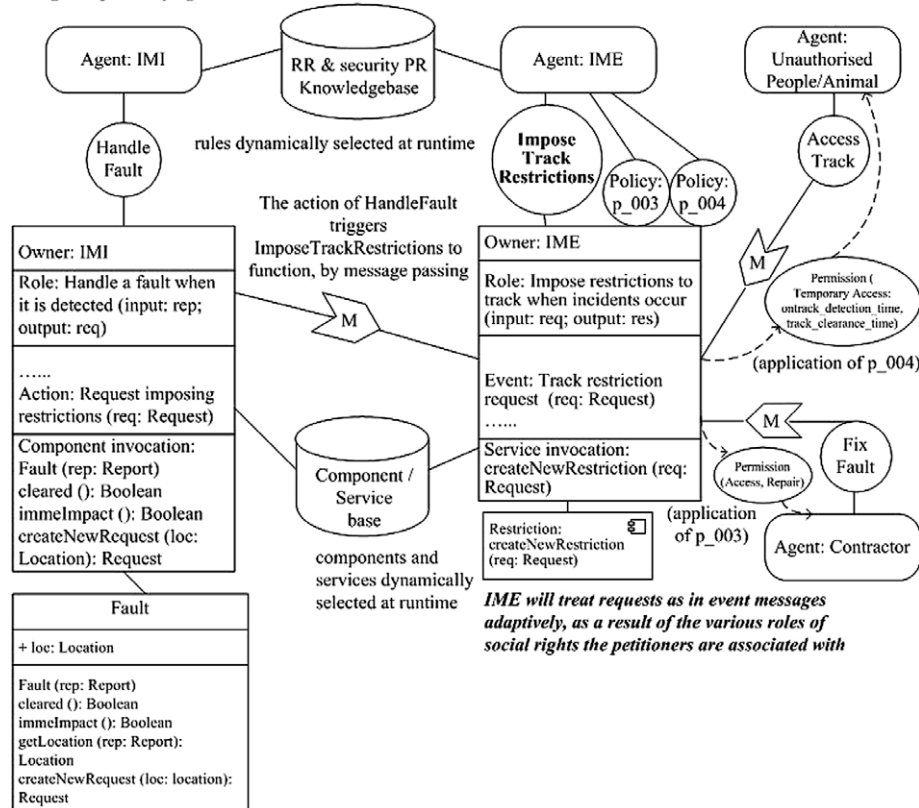


Fig. 22. Agents adaptively play functional duty role and comply with social right role.

tions among collaborative agents in the interaction model are then successively established towards a business process that gets enacted at runtime. In the business perspective, the business partnerships, business decision making, and business processes are all adaptive to changing requirements. The accurate relationships among these modelling elements for the example are shown in the lower section of Fig. 22.

It is also straightforward to define new interaction models to deal with new types of previously unforeseen business events, by setting new rules and assigning the appropriate components/services for a set of interactive agents. The configuration and execution of a new interaction model is carried out at runtime without interrupting the running system. As more and more interaction models come into being, the publication of them allows the integration of multiple interactions into a super-interaction model. For example, the train service reschedule interaction model could be part of the larger train service running interaction model involving both normal train running and amendments to it. The interaction model composition process can be assisted by a goal-oriented (de-)composition approach [32,35] by the AAM approach.

## 5.2. Adaptivity of social rights

Apart from the adaptation of functional interaction model, security policy rules can also be adapted, in a separate process, assign-

ing access rights to critical system data and services without affecting the core functionality. One may add/amend, according to the security meta-model, a security PR based on a specific subject, a role, or an organisation, affecting various numbers and types of subjects in their resource access model when bound with RR.

For example, when a Contractor requires examining infrastructure assets in addition to the tracks it is responsible for due to their interrelationship, a security PR should be defined to grant it such extra power. If another Contractor fails to repair a reported track fault in time and this needs immediate response then we may have to specify a security PR and allow an alternative Contractor to have access and do the repair operation in place of the original one. Supposing business mergers and collaborations occur, more train operators could come into play with more assets and train journeys coming under the management of the original system. We must then grant roles and organisations, previously unknown, the appropriate access to the existing resources. Moreover, existing roles may need access to additional resources previously seen as external. In many special occasions, a context can be used to cope with situations not predicted in advance. For example, we may allow a supervisor role from what is now an affiliate train company to review and adjust our timetable with a restriction by using a context component that defines trustworthiness.

Being retrieved and evaluated by agents at runtime and configured separately by human experts, RRs and PRs bring adaptivity to



a secure MAS system. It is only at the time that a subject requests access to resources that it knows what level of access it can have. The externally specified PRs, which are always up-to-date, are retrieved at the time the resource under protection is being requested. In this way, we can immediately impose more restrictive rules upon suspicious resource requesters and impose less restrictive rules upon those who indeed need resources and in whom we have trust.

### 5.3. Related modelling approaches towards easier (security) adaptation

Policies are a widely accepted form in which to capture requirements. Policy-based models can be transformed and deployed into running systems following the MDA philosophy. In one approach [39], policies are specified for the composition of role playing behaviour at the functionality level, evolving the original functional model to more a comprehensive behavioural model when needed. The use of policies, however, is limited to adding new functionalities, but not reconfiguring non-functional attributes. The extension of policy notions from capturing functional needs towards non-functional security needs will offer a security-enabled model-driven architecture.

A more generic software development model, the Twin Peaks Model [40], separates problem structure and specification from solution structure and specification. This model intertwines requirements and architectures under continuous and concurrent development, towards progressively more detailed specifications. Being an enhanced spiral life-cycle model, it claims to support the building of a stable yet adaptable architecture in the presence of changing requirements. In this respect, it is comparable with an agile development method which usually implies an incremental security architecture [41]. In our approach, the “twin peaks” of security model and functional interaction model under continuous maintenance are separate but built together, implying an equally decoupled incremental development process. However, the requirements are fine-grained and the architecture is actually driven by the continuously refined requirements. The result is, the maintenance of functional requirements is decoupled from the security concerns and the maintenance of them together is the maintenance of the actual running system. This has an advantage over the Twin Peaks Model in that the separate development of architecture from incrementally detailed requirements via an implicit link becomes unnecessary due to the establishment of a more directly connected path between problem and solution. Moreover, the incremental development of functional and non-functional requirements in independent processes makes them more easily manageable.

## 6. Contributions, conclusions, and future work

Role has been accredited importance in agent behaviour modelling for MAS. Also role has been central to permission assignment and management in access control. This paper offers an integrated role notion and, based on that, an integrated security-aware modelling approach in the paradigm of model-driven architecture. To the knowledge of author, no previous work has been carried out in associating the role of functional duty and the role of social right, though such a relationship is natural and of substantial value to directing new topics in both role-based security research in MAS and role-based security model development in MDA. Itself being an innovative approach towards agent-oriented security model-driven architecture, the work's main contributions to existing research communities are fourfold.

Firstly, AAM contributes a method of building adaptive and secure MAS with overall development process support to the agent-

oriented software engineering community. The AAM interaction model is constructed with associated behavioural semantics. The maintenance of the model is the maintenance of the system, agents always interpreting their behaviour from the model, which always has up-to-date requirements. The Agent-oriented MDA paradigm covers the complete MAS development process, filling the gap between major agent-oriented methodologies like *i\** or Gaia, which focus only on requirements and the early design phase, and agent-oriented development platforms such as JADE which focus only on the implementation phase. Very importantly, the security model add-on makes the systems developed from AAM more secure for use, which could boost the wider adoption of MAS in the software industry. This is further facilitated by the use of business-friendly security policy rules, which give business experts the full configurability of their security needs and have already been in use in one form or another in large-scale OO systems.

Second, AAM provides to model-driven architecture researchers a means of abstracting high level business-oriented model, understandable by business people, so that they can bring real time change effects without caring about the implementation of low level computing constructs. The abstracted model is concerned with business knowledge including business partnership, business decision making, business policy as well as security policy associated with real business assets that are valued by business stakeholders. Business requirements formalised as such, they are easier to be maintained and validated at the model level. This model structuring is in contrast with UML (or Agent UML [22]), action semantics (AS) [23] or object constraint language (OCL) [8] where the notation system has to be manually interpreted by human beings during development, and in the case of OCL, being unable to capture high level business semantics. An important lesson learned from MDA experience is that models should be used to abstract selected elements of the implemented complex systems rather than replicate the abstractions in the programming languages [24]. AMDA raises the level of abstraction from MDA's low level object and object constraint concepts to business-oriented constructs, capturing high level interaction, decision making, policy application, and so on.

Associated with this shift in level of abstraction, the characterised approach further directs a means to tackle the growing issue of software complexity [25]. Traditionally, the inherent technical complexity of systems is overcome by its decomposition into smaller chunks in size and then statically assembling the whole from the parts. A tougher type of complexity is now being recognised concerning the dynamic nature that allows their entities to individually perform dynamically or collectively interact adaptively under various social contexts. An analogy is that a computer can be used to run an application. Although its components of CPU or memory can be upgraded physically and reconfigured internally by technicians, the computer's social role of running applications is maintained without change of its social position to its user, even though the user could feel the application now runs quicker. The functionality is largely separated from social role in AAM and this enormously reduces social complexity. Agents abstract at the business and social level the role duties and role rights constraints, respectively. The interaction model captures interactions of these socially and the internal functions of agents provided by components or services are separated below the business and social level. The layered architecture not only separates the underpinning function facilities and the social activities that can make use of the available facilities, but also provides a means of configuring this from a business point of view.

Third, AAM contributes the extension of MDA to distributed computing and also enables distributed systems to take advantage of the MDA paradigm. We allow in our model, the use of low level computing facilities developed not locally for a single closed sys-

tem, but disparate components and services in a distributed environment. This enables AAM to cope with the challenges brought by the pervasive business settings and the associated so-called global requirements engineering [26]. This is a non-trivial feature of AAM since software is no longer a monolithic system running on a single computer [25] but rather evolves in line with business acquisition and collaboration, making use of heterogeneous services. Components and services may be ready to use but in a context their original developers have not planned, their integration from various sources in an open environment such as Internet being an emergent need. AAM agents, being technology-independent, are well suited to the coordination and interoperation [34] of separately developed components and services in our adaptive model. Their individual use or interaction among them is not coded in advance so that emerging business needs can be met via selection and connection of the relevant ones at runtime.

Fourth, AAM contributes a model-driven structure to the Role-Based Access Control community. The role based security model is extended and policy rules defined with, instead of two, four dimensions with richer configurability such as: access subjects can be specified on the basis of role as well as individual and organisation; access objects can be data as well as service; access operations depend on specific data/service operations; and contexts provide extra flexibility. The integration of the security model into the AAM framework allows agents to dynamically evaluate and apply the appropriate policies before they perform their actual capabilities, a behaviour being driven by the combinational model. But the change to security needs has no effect on core system functionalities. This allows the definition of any number of policies after the system has been developed, security requirements not being tangled with others. The RBAC research can benefit this new model-driven security model.

We may, in our future work, develop more powerful self-adaptation features and include it into the framework. The aim is to let agents discover by themselves insecure or unreliable service providers as well as service requester and then automatically replace or deny them to maximise the security and performance of systems autonomously. The sense and applicability of this, however, demands further investigation. Sometimes human experts are still the best source that tells right from wrong and a fully automatic system is usually hard to achieve without some trade-off in accuracy, consistency or other desirable attribute. For example, if a Contractor is found to always fail to provide a satisfactory fault repair service can we rely purely on agents to automatically replace it with a reliable service provider? If unauthorised (animal) access is always reported from a particular part of the track, can the system address this in the way as if a hacker attempts to break in the system, or do we have to send someone to the site and physically find what is wrong? Nevertheless, adding autonomy to the existing security-aware agent-oriented MDA paradigm provides an interesting direction for further research.

In the interest of concentration on security concern modelling and its integration into the existing AAM framework, we have ignored the discussion of the model interpretation engine, which is a prerequisite of the architecture that drives constrained agent behaviour. For a more complete view of the AAM approach in this aspect, please refer to [29], which provides a general illustration of agent-oriented MDA paradigm, without security concern. A blueprint diagram of the PIM including RR and PR as well as other model elements have been put together that guides the later development work. In our future work, the existing framework that already accommodates security aspect will offer fully automatic model transformation capabilities. Work in this direction is underway. In recent work, the supporting requirements engineering method used in the beginning phases of the modelling approach has been presented [35]. The complete knowledge

hierarchy originating from requirements and including RR and PR is described in [31]. We have implemented an e-commerce prototype system in the JADE environment [12] using the approach with tool support [30,32] and given evidence for its adaptivity quality. An architectural deployment view of the approach is given in [27 and 28]. The use of the approach in achieving adaptivity in distributed environment is discussed in [34]. The preliminary security model [33] that this work is based on has now been elaborated and extended in this paper.

Overall, providing security in MAS is still in need of elaboration and new methods and tools need further development. Meanwhile, the approach presented here offers an advance towards a practical method of developing secure MAS which are adaptive both in terms of behaviour and security needs.

## Acknowledgement

Thanks to Des Greer for helpful comments and suggestions for improving the paper.

## References

- [1] J. Jurjens, M. Lehrhuber, G. Wimmel, Model-based design and analysis of permission-based security, in: Proceedings of the 10th IEEE international Conference on Engineering of Complex Computer Systems (ICECCS'05), IEEE Computer Society, 2005, pp. 224–233.
- [2] H. Mouratidis, J. Jurjens, J. Fox, Towards a comprehensive framework for secure systems development, in: Proceedings of the 18th International Conference on Advanced Information Systems Engineering (CAISE'06), Springer, 2006, pp. 48–62.
- [3] D. Kim, I. Ray, R. France, N. Li, Modeling role-based access control using parameterized UML models, in: Proceedings of the Seventh Conference on Fundamental Approaches to Software Engineering, Springer, 2004, pp. 180–193.
- [4] T. Lodderstedt, D. Basin, J. Doser, SecureUML: a UML-based modeling language for model-driven security, in: Proceedings of the Fifth International Conference on the Unified Modeling Language, Springer, 2002, pp. 426–441.
- [5] J. Jurjens, UMLsec: extending UML for secure systems development, in: Proceedings of the Fifth International Conference on the Unified Modeling Language, Springer, 2002, pp. 412–425.
- [6] A. Rodriguez, E. Fernandez-Medina, M. Piattini, A BPMN extension for the modeling of security requirement in business processes, IJCE Transactions on Information and Systems E90-D (4) (2007) 745–752.
- [7] R. Villarroel, E. Fernández-Medina, M. Piattini, J. Trujillo, A UML 2.0/OCL extension for designing secure data warehouses, Journal of Research and Practice in Information Technology 38 (1) (2006).
- [8] Object Management Group, 250 First Avenue, Suite 100, Needham, MA 02494, USA.
- [9] A. Kleppe, J. Warmer, W. Bast, MDA Explained: The Model Driven Architecture: Practice and Promise, Addison-Wesley, 2003.
- [10] Foundation for Intelligent Physical Agents. Available from: <<http://www.fipa.org/>>.
- [11] G.A.S. Torrellas, L.B. Sheremetov, An authentication protocol for agent platform security manager, in: Proceedings of the Emerging Technologies and Factory Automation, 2003, pp. 623–628.
- [12] Java Agent DEvelopment Framework, Available from: <<http://jade.tilab.com/>>.
- [13] A. Poggi, G. Rimassa, M. Tomaiuolo, Multi-user and security support for multi-agent systems, in: Proceedings of WOA, 2001.
- [14] S. Poslad, M. Calisti, Towards improved trust and security in FIPA agent platforms, in: Proceedings of the Autonomous Agents, 2000.
- [15] C. Farkas, M.N. Huhns, Making agents secure on the semantic web, IEEE Internet Computing 6 (6) (2002) 76–79.
- [16] G. Vigna, Mobile Agents and Security, LNCS, vol. 1419, Springer, 1998.
- [17] JADE Board, JADE Security Guide, 2005.
- [18] C. Petrie, C. Bussler, Service agents and virtual enterprises: a survey, IEEE Internet Computing 7 (4) (2003) 68–78.
- [19] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman, Role-based access control models, IEEE Computer 29 (2) (1996) 38–47.
- [20] J. Odell, M. Nodine, R. Levy, A metamodel for agents, roles, and groups, in: Proceedings of the Fifth International Workshop on Agent-Oriented Software Engineering (AOSE'04), Springer, 2005, pp. 78–92.
- [21] F. Zambonelli, N.R. Jennings, M.J. Wooldridge, Developing multiagent systems: the Gaia methodology, ACM Transactions on Software Engineering and Methodology 12 (3) (2003) 317–370.
- [22] AUML web site, Available from: <<http://www.auml.org/>>.
- [23] Object Management Group, OMG Unified Modeling Language Specification (Action Semantics), OMG document ptc/02-01-09, 2002.
- [24] R. France, S. Ghosh, T. Trong, Model-driven development using UML 2.0: promises and pitfalls, IEEE Computer 39 (2) (2006) 59–66.

- [25] J.L. Fiadeiro, Designing for software's social complexity, *IEEE Computer* 40 (1) (2007) 34–39.
- [26] D. Damian, Stakeholders in global requirements engineering: lessons learned from practice, *IEEE Software* 24 (2) (2007) 21–27.
- [27] L. Xiao, D. Greer, The agent-rule-class framework for multi-agent systems, *International Journal of Multiagent and Grid Systems* 2 (4) (2006) 325–351.
- [28] L. Xiao, D. Greer, Externalisation and adaptation of multi-agent system behaviour, in: K. Siau (Ed.), *Advanced Topics in Database Research*, Idea Group, vol. 5, 2006, pp. 148–169.
- [29] L. Xiao, D. Greer, Towards agent-oriented model-driven architecture, *European Journal of Information Systems* 16 (4) (2007) 390–406.
- [30] L. Xiao, D. Greer, Adaptive agents in a configurable environment, *International Journal of Multiagent and Grid Systems*, in press.
- [31] L. Xiao, D. Greer, A hierarchical agent-oriented knowledge model for multi-agent systems, in: *Proceedings of the Eighteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'06)*, 2006, pp. 651–656.
- [32] L. Xiao, *The Adaptive Agent Model*, Ph.D. thesis, Queen's University Belfast, 2006.
- [33] L. Xiao, A. Peet, P. Lewis, S. Dasmahapatra, C. Saez, M. Croitoru, J. Vicente, H. Gonzalez-Valez, M. Lluch, An adaptive security model for multi-agent systems and application to a clinical trials environment, in: *Proceedings 31st IEEE Annual International Computer Software and Applications Conference*, IEEE Press, 2007, pp. 261–266.
- [34] L. Xiao, D. Robertson, M. Croitoru, P. Lewis, S. Dashmapatra, D. Dupplaw, B. Hu, Adaptive agent model: an agent interaction and computation model, in: *Proceedings of the 31st IEEE Annual International Computer Software and Applications Conference*, IEEE Press, 2007, pp. 153–158.
- [35] L. Xiao, D. Greer, Agent-oriented requirements modelling, in: *Proceedings of the First International Workshop on Requirements Engineering for Business Need and IT Alignment (REBNITA'05)*, 2005, pp. 28–37, In conjunction with the Thirteenth IEEE Requirements Engineering Conference (RE'05).
- [36] A. Kern, A. Schaad, J.D. Moffett, An administration concept for the enterprise role-based access control model, in: *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies (SACMAT'03)*, ACM Press, 2003.
- [37] A. Korthaus, Using UML for business object based systems modelling, in: M. Schader, A. Korthaus (Eds.), *The Unified Modeling Language – Technical Aspects and Applications*, Physica-Verlag, 1998, pp. 220–237.
- [38] J.T. Pollock, R. Hodgson, *Adaptive Information: Improving Business Through Semantic Interoperability, Grid Computing, and Enterprise Integration*, Wiley-Interscience, 2004.
- [39] J. Pena, M.G. Hinchey, R. Sterritt, A. Ruiz-Cortes, M. Resinas, A model-driven architecture approach for modeling, specifying and deploying policies in autonomous and autonomic systems, in: *Proceedings of the Second IEEE International Symposium on Dependable, Autonomic and Secure Computing*, 2006, pp. 19–30.
- [40] B. Nuseibeh, Weaving together requirements and architectures, *Computer* 34 (3) (2001) 115–117.
- [41] H. Chivers, R.F. Paige, X. Ge, Agile security using an incremental security architecture, in: *Proceeding of the Sixth International Conference on eXtreme Programming and Agile Processes in Software Engineering*, LNCS, vol. 3556, Springer, 2007, pp. 57–65.
- [42] E. Fernández-Medina, J. Trujillo, R. Villarroel, M. Piattini, Access control and audit model for the multidimensional modeling of data warehouses, *Decision Support Systems* 42 (2006) 1270–1289.