Author(s): Beydoun, Ghassan; Low, G; Mouratidis, Haralambos; Henderson-Sellers, Brian.
Article title: A Security-Aware Metamodel for Multi-Agent Systems (MAS)
Year of publication: 2009
Citation: Beydoun, G. et al (2009) ‘A Security-Aware Metamodel for Multi-Agent Systems (MAS)’ Information and Software Technology 51 (5) 832-845
Link to published version: http://dx.doi.org/10.1016/j.infsof.2008.05.003
DOI: 10.1016/j.infsof.2008.05.003

A Security-Aware Metamodel for Multi-Agent Systems (MAS)

Beydoun[1], G., Low[2], G., Mouratidis[3], H. and Henderson-Sellers[4], B.

[1]University of Wollongong, Wollongong, Australia
beydoun@uow.edu.au
[2]University of New South Wales, Sydney, Australia
g.low@unsw.edu.au
[3] University of East London, England
haris@uel.ac.uk
[4]University of Technology, Sydney, Australia
brian@it.uts.edu.au

Abstract. This paper adopts a Model Based Security (MBS) approach to identify security requirements during the early stages of multi-agent system development. Our adopted MBS approach is underpinned by a metamodel independent of any specific methodology. It allows for security considerations to be embedded within any situated agent methodology which then prescribes security considerations within its work products. Using a standard model-driven engineering (MDE) approach, these work products are initially constructed as high abstraction models and then transformed into more precise models until code-specific models can be produced. A multi-agent system case study is used to illustrate the applicability of the proposed security-aware metamodel.

Keywords: Security-based Metamodelling, Model Based Security, Modelling of Secure Agent-Oriented Information Systems, Modelling of Security features, Peer-to-Peer Security Modelling.

1. Introduction

In the context of conceptual modelling and model-driven software engineering, (software) agents can be defined as conceptual entities that exhibit autonomy, situatedness and interactivity. They are situated in an environment in which they are able to sense and respond to changes. Agents have been found useful in model-based development of open, distributed and heterogeneous systems. However, as has been argued in the literature [23], for agent technology to be widely recognized, the security issues that surround agents must be resolved.

Research efforts, so far, have mainly focussed on solving individual security problems of multi-agent systems (MAS), such as attacks by an agent on another agent, attacks from a platform on an agent, and/or attacks from an agent on a platform [14]. Security is not yet considered as part of the development process of a MAS. This is partly because existing methodologies, modelling languages and methods for the development of MASs do not generally incorporate abstractions and processes that support the consideration of security issues. Rather security is often considered only after the design of the system has been finalised, which leads to various security vulnerabilities [19].

In this paper, we produce a methodology-independent security (meta) model that can be used in the construction of any situated methodology [7] as required by the context of the MAS development project. It is aligned with Model Based Security (MBS) as proposed in [16] as a means of supporting the consideration of security from the early stages of the information system development process. Initially, high abstraction models are constructed and transformed, following a standard model-driven engineering (MDE) approach, into more precise models until code-specific models can be produced. We believe this approach, combining agents and security in an MDE context, can be successfully employed in the overall development of a multi-agent software system. Towards this, this paper provides the foundations for the construction of the models in the form of an agent-oriented modelling language that incorporates security considerations.

We present a MAS metamodel that defines security concepts along with agent development concepts. Our MAS metamodel described in this paper has the capacity to model the security requirements of any given MAS independently of the process used to create it. It is based on the FAML (FAME[1] Agent-oriented Modelling Language) generic metamodel [1] and previous work on security-aware agent metamodels [2]. The chosen security concepts are designated into two sets: run-time concepts and design-time concepts. Each set has two scopes: system-related or agent internals-related scope. Our work is part of a greater effort to develop secure multi-agent systems, based on the application of model-based security and a conceptual modelling approach to address security requirements of multi-agent systems, as suggested in recent work e.g. [10, 19]. This allows developers to account for security of a MAS early during the development of the system rather than as a costly afterthought.

The rest of this paper is organized as follows: Section 2 describes related work. Section 3 briefly outlines the FAML metamodel and the analytic process used to identify the required security modelling units (classes in the metamodel). Section 4 articulates the MAS-specific security concerns of any MAS and associates these with basic modelling primitives. Section 5 incorporates these primitives into our metamodel and extends the metamodel to accommodate all security concerns identified in Section 4. Section 6 illustrates the semantics of the modelling units of our metamodel on a P2P community sharing application illustrating our model driven

[1] FAME (Framework for Agent-oriented Method Engineering) is the project name under which FAML has been developed.

approach to develop a securitised MAS. Finally, Section 7 concludes with a discussion of future work.

2. Modelling and Developing Secure Agent-Oriented Systems

The term agent derives from the present particle of the Latin verb agere, which means to drive, act, lead or do [5]. Although there is no standard definition of what a software agent is, it is widely agreed that, in broad terms, an agent demonstrates the following properties: (i) Autonomy. Agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state; (ii) Social ability. Agents interact with other agents (and possibly humans) via some kind of agent communication language; (iii) Reactivity. Agents perceive their environment, (which may be the physical world, such as a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it; (iv) Pro-activeness. Agents do not simply act in response to their environment; they are able to exhibit goal-directed behaviour by taking the initiative.

Work within the agent research community has led towards the development of agent-oriented software engineering (AOSE) paradigm. AOSE introduces an alternative approach in analysing and designing complex distributed computerised systems [13, 15, 36], according to which a complex computerised system is viewed as a multi-agent system (MAS) [13] in which a collection of autonomous software agents (subsystems) interact with each other in order to satisfy their design objectives. Therefore, when developing a MAS, this can be viewed as a society, similar to a human society, consisting of entities that possess characteristics similar to humans such as mobility, intelligence and the capability of communicating [13]. To assist the development of multi-agent systems, a number of methodologies (see for instance Tropos, GAIA, MaSE) and their associated process-focused metamodels, as well as modelling languages (see for instance AgentUML, AML) have been proposed. In this paper, we focus on metamodels for the atomic paradigmatic element (here, agents) and its associated work products and do not discuss any further process- or method-focussed metamodels such as the OMG's SPEM or the International Standard ISO/IEC 24744.

Although security has been identified as an important issue [23] for the widespread use of agent technology, most of the methodologies and modelling languages either ignore the security aspects related to MAS development or only provide partial treatment of security concerns. We briefly review here the literature that considers the security issues for multi-agent systems.

Liu *et al.* [18] identify security requirements during the development of multi-agent systems, in which security requirements are analysed as relationships amongst strategic actors, such as users, stakeholders and potential attackers. These authors propose three different kinds of analysis techniques: agent-oriented, goal-oriented and scenario-based analysis. Agent-oriented analysis is used to model potential threats and security measures, whereas goal-oriented analysis is employed for the development of a catalogue to support the identification of the different security relationships within the system. Finally, scenario-based analysis is considered as an elaboration of the other two kinds of analysis.

Secure Tropos [21] is an extension to Tropos [6] and is the first methodology to consider security issues during the development of multi-agent systems, extending Tropos with security-related concepts and introducing a security-related process that allows developers to identify the security requirements of a multi-agent system, to transform these requirements into a design and to test the developed system against a number of potential security attacks [21]. Huget [12] proposed a new agent-oriented methodology, called Nemo, and claims it tackles security. From the current description of the methodology, security seems to be considered only superficially with security not considered as a specific model but included within the other models of the methodology. The developer even states "security has to be intertwined more deeply within models" [12]. Therefore, more evidence is required to satisfy the developer's claim that the methodology tackles security.

To the best of our knowledge, there has been, to date, little or no effort to provide a model-based security approach for multi-agent systems. Although a number of works support the concept of model-based security for software systems (e.g. [16, 22]); these works do not adequately support the development of multi-agent systems since they omit important agent-oriented concepts and abstractions. Hence, developers find no help when considering security during the development of MAS and the common approach towards the inclusion of security within an agent-oriented system is to identify security requirements after the definition of a system [23]. This typically means that security enforcement mechanisms have to be retrofitted into a pre-existing design. This approach leads to serious design challenges that usually translate into the emergence of agent-based systems afflicted with security vulnerabilities [4, 23].

Work has commenced on agent-oriented modelling languages that could support a model-based approach for the development of multi-agent systems. The Agent Unified Modelling Language (AUML) [25] is an extension of the well-known UML aiming to model agent systems. Its support for agent-oriented concepts is heterogeneous although there are some extensions that accommodate the distinctive characteristics of agent systems such as autonomy and mobility. However, AUML does not consider security issues.

3. The FAME Agent-oriented Modelling Language

Within the overall framework of the FAME (Framework for Agent-oriented Method Engineering) project, we have identified the need to include a modelling language. Such a modelling language defines concepts from which can be instantiated modelling elements from which a model (a design) can be constructed. The design can then be hand-coded or used as the input to model-based (or model-driven) information systems development, as in MDE (model-driven engineering) or a specific flavour of MDE like OMG's Model-Driven Architecture (MDA) [17, 27]. This modelling language was developed as a generic approach by the study of a large number of specialized and highly focussed (sometimes linked to a single methodology) modelling suites, some of which included a metamodel description. The resulting FAML (FAME Modelling Language) is essentially a metamodel, depicted typically using UML [27, 28]. The 4-step, iterative research methodology we adopted to ensure quality and genericity of this model is detailed in [1] and the results summarized in this section.

It was soon realized that a single interconnected metamodel would be unusable in practice and that, instead, we identified two dimensions along which a more rational metamodelling result could be portrayed. Firstly, in agent-oriented software modelling, there is a rough discrimination between design time and run time – in an MDE context this reflects to some extent the difference between the PIM (platform independent model) and the PSM (platform specific model) [26]. Secondly, there are two viewpoints that can be termed agent internal and system focussed. This leads to four distinct diagrammatic groupings (Fig. 1). These four definitional views, as published in [1] are shown in Figs. 2-5.

**** Figs. 1-5 about here ****

Fig. 2 shows the classes of the metamodel that are directly related to the description of design-time system-related classes. These are concerned with features that can only be perceived by looking at the whole system at design time. Fig. 3 shows the classes related to the environment in which agents "live", that is, run-time environment-related classes. Fig. 4 shows the classes related to the agent internals at design time and Fig. 5 shows the classes related to agent internals at run-time, classes that can only be perceived by considering the internals of agents at run-time. The ontological definitions for design-time and run-time concepts are given in Tables 1 and 2 respectively.

**** Table 1-2 about here ****

The combination of these four diagrams, which is the FAML definition, was evaluated in [1] by comparison with TAO [33] and Islander [8]. However, none of these approaches (FAML, TAO, Islander) deal with security issues. Consequently, we have introduced security concerns into the existing FAML metamodel as described below.

4. Security Requirements of a MAS

In this section, we analyse the security requirement of multi agent systems to extend FAML to support modelling of security concerns. The extension of FAML consists of two sets of modelling classes (metaclasses): One set to model the security requirements of a multi-agent system (MAS) identified by the developer during the design stage; and another to model security actions satisfying the security requirements. These modelling classes will be added to the four views of the FAML metamodel shown in Figs. 2-5 to produce the new version shown in Figs 6-9. Our security-aware MAS metamodel (Figs 6-9) enables developers to consider security vulnerabilities of individual agents at all stages in a model-driven development project. We argue that security requirements of a multi-agent system can be generally categorized into two types: System Security Requirements (SSRs) and Agent Specific Security Requirements (ASSRs). SSRs are used to model security requirements that apply to the system as a whole, i.e. that all the agents of the system must satisfy, whilst ASSRs are used to model security requirements that only apply to individual agents.

The analysis in this section runs along two dimensions: the dimension of increasing complexity of security risk (see Section 4.1) and the dimension of inherent attributes of agents in a MAS. The latter dimension includes the agent inherent attributes: autonomy, mobility and cooperation (see Sections 4.2-4.3). Modelling unit sets associated with these attributes may overlap.

4.1. Agent-specific vulnerability levels

The first dimension of analysis, the security context complexity, ranges from a single agent on a single machine to a fully mobile MAS with agents capable of roaming the net using dynamic multi-hop routing. Although a grey scale, we can characterize an agent that carries its code, data and its state of execution (i.e. program counter and CPU registers) as a *strongly* mobile agent and an agent that only carries code and data as being *weakly* mobile. We start our analysis in the least risky environment (a single agent on a single machine) although we aim to extend this later in order to overcome its limitations in increasingly risky settings until we have a complete security model that can represent all security requirements and solutions for the riskiest setting.

In total, we identified five different levels of vulnerabilities as shown in Table 3. The agent vulnerability level and, consequently, the need for protection such as encryption, increases as we move from Level 0 to Level 4. In the case of Level 2, data are transported (for message passing) while, for Levels 3 and 4, both data and code may be transported (for remote evaluation), thus allowing application level attacks. Level 4 is considered more vulnerable than Level 3 due to the additional complexity of the agent also carrying its state of execution (i.e. program counter and CPU registers). Within Levels 3 and 4, the mobility of the MAS is important. For instance the level of distribution may range from a single hop, to multiple fixed hops, to dynamic multiple hops. As the number of hops and the degree of freedom increases, it becomes more complex to ensure safe transportation when intermediate co-operation is not guaranteed (hostile hosts may be encountered – this is not the case for distributed systems). The scale of vulnerabilities, as shown in Table 3, delineates the research issues at each level of complexity and can be used as a research roadmap to fulfil MAS-specific security requirements of the most complex forms of MAS.

**** Table 3 about here ****

Agent security requirements for communication channels are equivalent to the normal requirements for confidentiality, integrity, authentication and availability required by a typical software application [4]. Security of communication channels is not included in the MAS-specific security requirements. Our security framework strives for authenticated communication between agents (including mobile agents), where any receiving agent can ascertain the identity of the sender and can choose to block an agent if it does not want to interact with it. Moreover, any agent can deny/offer access to its owned resources (including its internal state) at any time it wishes. Resources that need to be protected can be local (owned by a single agent) or global (owned by many agents), be distributed or reside on a single device.

4.2. Co-operation, autonomy and security of agents

A MAS can offer new services and functionality created from a combination of specialized services of individual agents. Typically, this requires co-operation between agents in order to make MASs adaptive and versatile when encountering unforeseen problems. It is therefore important to be able to maintain co-operation between agents by ensuring that interactions that exist between agents specifically in order to share resources are kept secure. Restricting access can hinder functionality and is not always a viable option. To simplify our analysis, in this section we assume that the agent's resources are localized with the agent and that any resource is owned by only one agent.

Co-operation between agents in a MAS requires mutual agreement between agents to share resources and, in some instances, to share access to their internal states [35]. The broader the mutually agreed access to resources and internal states is, the richer the potential functionality of the system. It is essential for agents to be able to trust other agents in order to provide a broad spectrum of interactions. In order to implement a mutual agreement to share access to resources and internal states between agents, i.e. to co-operate, it is critical for agents to discern changes in their states or resource arising both from a legitimate access and from changes arising from malicious intrusion. A way to achieve this is for each message to be uniquely associated with an agent identity and time-stamped. We include *agent_identity* as a basic modelling unit to describe an agent uniquely. This identity can be a function of the system to which an agent belongs; although in the case of open systems we note that this would not be important. This serves as a fingerprint of any agent created in any MAS. A higher level security concept, *signature,* can be implemented using *agent_identity*. A signature should be known by trusted parties and producible by the relevant identity [32]. The extended metamodel will also describe the relation between agents and their resources through *ownership* and *usage* relationships.

To maintain its autonomy, an agent should be resilient and able to recover from an interaction that gives access to unauthorized resources. In other words, maintaining some state description of its past interactions is required. Each agent should have an *interaction_history* log to allow for recovery. This log is only accessible to the agent. This concept along with *ownership* secures the internal state of agent. An agent within the system can then co-operate to reinstate its state as well as the state of the MAS if and when needed. Mediated systems e.g. [9] use a system interaction history, but this requires centralized access, which we avoid since it places limits on the mobility of agents. Our decentralized framework can model a mediated solution by having a single agent designated to mediate and to interact with all other agents. The interactions log of the 'mediator-designate' will then be a log of all interactions within the system, as in [9].

4.3. Mobility and security of agents

Mobility allows agents to replace remote procedure calls, saving on bandwidth and allowing computations not otherwise possible. In comparison with traditional distributed systems, mobility allows additional functionality for a MAS but also incurs additional security requirements. Mobility of agents varies, as noted in Table 3. For *weakly* mobile agents, discerning intrusion is easier, since the agent starts execution from scratch when it reaches its host. The essential problem with mobile agents is compounded by the fact that the host may require access to agent execution states [37]. Hence, if the host is malicious, intrusion recovery is needed.

Assuming that an agent is transmitted safely along a channel, threats to the resources of an agent (mobile or non-mobile) come from interacting with, *inter alia,* other agents, the host or users themselves. For example, there may be one or more mobile malicious agents designed to steal or corrupt data in the environment of the agent under consideration; the host that controls its execution could mistake the identity of the agent or it may simply be a malicious luring host; or unauthorized users might attempt to corrupt or steal an agent's data, or might attempt to infiltrate the functionality of the MAS.

A mobile agent system will support several networking protocols, which will allow it to transmit itself over a network. This could expose an agent to additional sources of threat through interacting with less trusted sources, the extent of the vulnerability of a mobile agent depending on the freedom it has with respect to its *mobility*. Mobility can be single hop (from host to another without any intermediate hosts) versus multiple hop (one or many intermediate hosts) or fixed (travel path is fixed and statically decided) versus dynamic (path is decided by agent as it travels – it is said to be roaming). Mobility requires the *location* modelling unit so that an agent is able to reason about its movements. Combined with its *interaction_history* (history of hops as well for mobile agents), it can reinstate itself into a safer location if it is under attack.

In the next section, we integrate all modelling concepts identified in this section into the existing FAML metamodel (Figs. 2-5).

5. Proposed Metamodel

This section presents the new version of FAML which accounts for security requirements. The new security concepts added to FAML are first presented (Table 4).

As these concepts are added, some existing FAML concepts are changed and some new non-security concepts are added (Table 5). The new 'securitised' FAML is presented in Figs. 6-9

As noted in the previous section, MAS-specific security requirements are of two kinds. The first kind refers to general security requirements that are not specific to one particular agent in the system. These requirements need to be accommodated by all agents. The second kind impacts individual agents within the system in different ways as mandated by the application. In Section 4, the focus was on the first kind and, more specifically, those that are engendered by the inherent characteristics of agents and MASs. This has resulted in the identification of a set of new modelling units to be integrated into the FAML metamodel. These are shown in Table 4 while their associations and attributes are shown in Figs. 6-9. As in the original FAML, we continue to differentiate between security requirements that are modelled by the software developer during the design stage and security actions that are performed by the multi-agent system during run-time in order to satisfy the security requirements. This allows for the development of a security-aware platform-independent design, providing part of a PIM for MDA. The realization of a working security-aware multi-agent system for a specific platform generated from a FAML instantiated MAS development methodology provides further support for a MDE development approach.

*** Table 4 here

At the system level, at design time, we view security requirements as part of the early design specification of the system. To support this, we introduce into the FAML metamodel the concept of security requirement, which we define as a security-related desirable property of the MAS that constrains its functional requirement(s). We distinguish between two classes of security requirements. Firstly, we identify system security requirements that relate to the integrity of the whole MAS. In other words, these are security requirements that must be fulfilled by all the agents of the system either individually by each agent of the system or through agent co-operation. We call these System Security Requirements and it is this aspect that has been our focus in this paper. We also add Agent-specific Security Requirements to denote security requirements that apply to an individual agent. Such security requirements, which are usually complementary to the system security requirements, define security-related properties that individual agents have to satisfy. Usually, in every multi-agent system there is an analogy/balance between system-specific and agent-specific security requirements. A high number of system security requirements imply a low number of agent-specific security requirements and vice versa.

A number of classes related to security and mobility were introduced into the metamodel. This resulted in a number of modifications in a number of classes of the original FAML metamodel. The modified or new (non-security) concepts are shown in Table 5 which provides a definition of new FAML and redefined classes. Associations and attributes of these concepts are shown in Figs. 6-9. For example, the new definition of the AgentDefinition now includes two new attributes (AgentIdentity and AgentType) and two more associations (Has and IsImposed) (see Fig. 8)..

*** Table 5 here ***

The extended FAML metamodel, including new concepts in Tables 4-5, is shown in four diagrams (Figs. 6-9) depicting the integration of security modelling units within the existing views: design-time system-related, runtime system-related (environment), design-time agent-internals and runtime agent-internals.

**** Fig. 6 about here ****

In FAML, the security framework is underpinned by recognizing and modelling the status of access to resources during development. The security requirements are modelled as a specialization of the Non-Functional Requirements. The security requirements are further specialized into system-specific security requirements (SSRs) and agent-specific security requirements (ASSRs) (see also Section 4). These security requirements add additional security goals and security tasks to the system goals and system tasks respectively (Fig. 6). To effectively implement its security requirement, we propose that a MAS views part of its resources as private, in order to protect and share only reservedly, and public in order to share more freely. Thus, in the proposed security extension, the modelling units Private Resource and Public Resource are added (Fig. 7). Private resources are agent-specific e.g. an agent's history log of hops as well as interactions with other mobile agents (Interaction_History).

**** Fig. 7 about here ****

In the agent definition-level at run-time, each agent assumes responsibility for its security. This is modelled with Recover Action Specification (Fig. 8). An example recovery action may be what the agent does to use the interaction log in order to reinstate its state and perhaps to assist in the reinstating of the MAS if an interaction has incorrectly given access to resources. This is a refinement of a more general modelling unit, Security Action Specification. FAML is extended with modelling units to express mobility behaviour of agents with the metaclasses Relocate Action Specification and Location Specification. Relocating is a restricted action that requires access to secured resources of the agents (i.e. only the agent can relocate itself).

**** Fig. 8 about here ****

At runtime, central to agent security is authentication, together with recovery for when authentication fails. Therefore at the agent-runtime scope (shown in Fig. 9), FAML is extended with the metamodel classes that represent the various kinds of resources and their access (the same as agent-design time taxonomy of resources). In addition, it is extended to permit modelling of restricted actions.

**** Fig. 9 about here ****

As derived from the above discussion, the language does not differentiate between different types of requirements and/or security solutions. Neither differentiates the source of the requirements. This means that security requirements identified through security policies and security requirements identified through other means (for cases where a security policy is not present) are effectively treated the same. If there are specific security policies, then the rules of the policies (depending on their type) initially are modelled with the aid of the design model (environment) as either system-wide or agent specific security requirements. Then, by employing the agent model (design), security constraints are derived. This process mainly depends on the methodology employed. For example, in secure Tropos security constraints model security requirements so the mapping is straightforward. Then, on the run time models, security actions are defined to represent possible security solutions for the identified security requirements. For instance, if the security policy of an organisation defines system-wide authorisation-related rules, these will be initially modelled as security requirements; security constraints will then be identified that enforce these requirements and security actions will be derived that provide security solutions that meet such security constraints and therefore the security requirements.

6. Illustration of the Metamodel on a MAS Application

As a practical illustration of security concepts in FAML, we consider a community-based search MAS application that we designed in [34]. This is a very complex application that involves most concepts of FAML. However, we will only give examples of the security-related concepts. We first describe the application and then show how FAML assists in identifying and modelling security issues.

6.1 P2P MAS application

Syntax-based search engines produce a very large number of hits most of which are irrelevant. They also overlook relevant webpages. Community-based search engines offer a promising alternative. The search engine keeps a decentralized track of users' common interests and history of queries to produce more accurate search results. Each human user is represented by an agent in the computer network to act on his/her behalf. This agent locates files and responds to queries from other similar agents. The collection of all these agents together with agents assisting them in their tasks form the P2P community-based-searching MAS. An agent representing the human user has access to a knowledge base containing electronic files that the user is willing to share with other users. Each file is identified by its title and type (e.g. HTML, pdf, music or video). As agents interact on behalf of their users, communities of interest begin to emerge. Agents develop an awareness of the communities to which users belong and use this awareness to fulfil their users' search requests efficiently and effectively, by interacting with the agents in the communities most likely to be able to serve their requests.

As human users pose a query to request files, the P2P MAS locates sites of other users where files matching the queries may reside, based on the querying behaviour of the users at those sites. The system mediates between human users, who are continuously represented by their local agents. An agent of another like-minded user may choose to respond to a query by providing details about the files it can supply, or by refusing the query. When all responses are received by the agent making the query, the agent combines and refines the results to compose a list of candidate files that satisfy the query. This is akin to a response from a web search engine but is shorter and more directly related to the query. The agent initiating the query can then select which file(s) it wants to download to its human user and it initiates a file transfer process with the agent who controls access to that file. Following a successful transfer, the knowledge base located where a particular query was made is updated to contain the received file(s) and to reflect the source of the file. For all agents involved in processing that query, their knowledge base is also updated with additional information reflecting the interests of the agent that initiated the query and further information about interests of other agents involved in the response. This information is used in future queries. In other words, as agents interact, they develop awareness of both the files possessed by their peers and which peers may be interested in the files that they themselves have.

At each node in the network, each user-agent keeps a record of its history of information sharing. The history contains two records: one of the past queries that it made on behalf of the human user and its respective responders, and one of the past queries received and their respective agent senders (acting on behalf of other human users). The former needs to be updated every time the user-agent receives a results list from the system, while the latter requires updating every time the user-agent replies to a query sent by the system. The history is used to produce short lists of candidate nodes for future queries, by calculating the similarity between the current query and a past query e.g. as suggested in [20]. If no nodes can be short-listed, or if no candidate user-agents provide the required service, the agent-user broadcasts the query to a wider circle of user-agents in the community to identify new candidate providers. In a fully evolved P2P system, agents may use their knowledge about other users' interests to request/negotiate for information from their peers when they do not know who has the files of interest. Any new providers are eventually added to the history, thereby expanding the user-agent's contact circle. The strategy of information sharing can be applied to any domain. The system is tuned to a domain using an external ontology describing the domain [3]. A fully deployed P2P community-based search system would have access to a suite of ontologies corresponding to various domains. As users use various ontologies to express their search, communities emerge. Details of how a community emerges or connects to another community (using a global ontology) are omitted here as they are not relevant for our description of security-related concepts.

6.2 FAML security concept examples in the P2P MAS application

In what follows, we illustrate the security aspects of FAML using the P2P application. FAML guides the identification of the P2P MAS security requirements. Accordingly, FAML's structure determines how they are identified and addressed according to its four views (shown in Figure 1): design-time system-related, runtime system-related (environment), design-time agent-internals and runtime agent-internals views. We first propose a set of the design-time system-related security concepts

(Figure 6). For each subsequent view (Figures 7, 8 and 9), we give example concepts refining the design-time security concepts.

6.2.1. Design-time system concerns in the P2P application

Security requirements can either be System Security Requirements or Agent Specific Requirements (see Figure 6). From the application description, we identify Agent Specific Security Requirements (ASSR):

1. An agent guards the history of queries it receives and sends.
2. An agent is able to guard its own identity.
3. An agent can prove its own identity.
4. An agent guards the identity of communities to which it belongs.
5. An agent guards resources (files) it owns for the purpose of sharing.
6. If an agent is a community gate keeper, it guards the membership to the community.

The above six requirements suggest the following System Security Requirements (SSR) of the P2P community-based searching system.

On the receiver side:

7. A sender agent's identity is authenticated before processing a search request.
8. A file is accepted from identified and authenticated senders who confirm and authenticate their identity at the start and at the conclusion of a transfer.
9. A sender agent's identity is confirmed before responding to a history query.

On the sender side:

10. Every search query sent is authenticated.
11. A file is transferred to identified and authenticated receivers who confirm and authenticate their identity at the start and at the end of a successful transfer.
12. Every history query that is sent is authenticated.

Requirements 7, 8 and 9 on the receiver side are equivalent to requirements 10, 11 and 12 on the sender side respectively.

The preceding 12 Security Requirements *satisfy* the following Security Goals (Figure 6) which we identify as follows:

1. An agent shares a file only when an explicit request is made by a known trusted agent or a member a known community. This goal satisfies requirements 8 and 11.
2. An agent shares data from its interaction history only when an explicit request is made by a known trusted peer or a member of a known community. This goal satisfies requirement 1.
3. An agent is the sole entity with access to its interaction history. This goal satisfies requirements 9, 12 and 1.
4. An agent is the sole entity that can search its files. This goal satisfies requirements 7, 10 and 5.
5. An agent has a unique identity identifier that can be securely transmitted to other agents if the agent decides to do so. This identifier can be changed only by the agent. This goal satisfies requirements 2 and 3.

6. An agent keeps track of communities to which it belongs. This goal satisfies requirement 4.
7. An agent can make an authentic request to community gate keepers regarding peer memberships. This goal satisfies requirement 4.
8. An agent can make an authentic request to community gate keepers to join a given community. This goal satisfies requirement 6.

As shown in Figure 6, according to FAML, Security Goals are achieved by Security Tasks. The above 8 security goals are achieved by one or more of the following Tasks:

1. Confirm an agent's identity against known peers.
2. Confirm an agent's community membership.
3. Maintain a list of community memberships.
4. Retrieve historical data.
5. Update interaction history.
6. Transmit own identity.
7. Request to become member of a community.
8. Nominate to become a community gate keeper.
9. Request identity of community gate keeper.
10. Broadcast new members to corresponding communities (by gate keeper).

The preceding 12 security requirements, 8 security goals and 10 security tasks form the set of system-related security classes at design time. In the rest of this section, we show examples of the security-related concepts for the remaining three views of the securitised FAML (as illustrated in Figures 7, 8 and 9).

6.2.2. Run-time environment concerns in the P2P application

Agent specific Security Requirements define Security Constraints at the environment level view (agent-external, runtime) shown in Figure 7. For example, consider Agent Specific Security Requirements 1, that an "agent guards the history of queries it receives and sends". This Agent Specific Requirement gives rise to the following Security Constraints:

- Queries involving private resources are authenticated.
- Community membership is known and shared.

The Environment-level view (agent-external, runtime) shown in Figure 7 indicates that Security Tasks have Security Action Specifications. For example, security Task 2 suggests the following Security Action Specification:

- Check a given identity against a known list of identities of agents. If the check is successful then perform a history query.

Another example of Security Action Specification is shown in Section 6.3.

An example of Private Resource Specification (in Figure 7) is:

- History of queries is a private resource specified as a database design storing all information about past queries. Every agent has such a database that it uses to guide its search queries and it shares this database with other trusted agents.

6.2.3. Agent internal scope in the P2P application

Examples of an Agent Private Resource in the FAML view of agent internals at design time (Figure 8) include:

- History of queries.
- History of agents contacted and details of contact.
- Files to be shared.

These Private Resources are *used* by an agent in the FAML view of agent-internals at runtime (Figure 9).

Other examples in the FAML view of agent internals at design time are:

- Recover Action at runtime is derived from the execution of the recovery action depending on the prevailing environment conditions applicable. For example the recovery in the case of a failed identity check (described in Action Specification 1 shown in Section 6.2.2) is different from the recovery in the case of failed file transfer (shown in Section 6.3).
- Location Specification: For example host locality and communities' membership details.

6.3. P2P application case study discussion

In our FAML-driven security analysis in Section 6.2, if a concept appears in more than one view, we gave examples of the first occurrence. Whilst we have presented the case study in a top-down fashion (from system-related security model units at design time to agent model units at runtime), it is important to note that this may not capture all the security-related components because there are non-security Tasks (at the system level view) that may involve Security Actions at the agent level view. For example if a file transfer is interrupted, the following more elaborate example of a Security Action Specification applies:

- Recovery from wrongly authenticated file transfer:
  - If transfer is not complete, abort file transfer and undertake a minor clean-up operation of partially transferred data.
  - If transfer is already finished, undertake a major clean-up operation of file, checking for any malicious code included in the file and in any other file involving any agents in the security breach.
  - Trace any side-effects including any pending requests, any modifications required to history, any modifications to security knowledge base.
  - Check for any past bad file from same source.

Within the same view, a non-security Task may also require an agent specific requirement. For example, at the System level view, community portal agents are appointed based on a request they make and confirmation by sufficient number of peer agents. The data stored in the history of interactions will form the basis of an agent volunteering and the subsequent peer confirmations. This is not security-related requirement, however, since it does involve security-related Tasks because an agent needs to make an authentic request to peers it has interacted with in order to form a community.

When the full model has been derived by analysis of all 20 security requirements in this P2P case study, and the independent analysis and verification of all security modelling units (at the other views in FAML) have been undertaken, the resultant model then forms the input to the model-based software development project. Since each element in the security-based agent model/design is conformant to an element in the MAS metamodel, it is defined unambiguously and precisely. Thus such a security-based design is an ideal input to an MDE style of application development (a description of the basic ideas underlying MDE can be found in e.g. [17]).

7. Discussion, Conclusions and Future Work

In this paper, we have presented work that provides the foundation towards a model-based security approach for the development of secure multi-agent systems. In particular, we have extended the FAME Modelling Language (FAML) and we have defined a metamodel that supports the development of security models for agent-oriented systems. The original FAML metamodel [1] did not accommodate the security requirements of the system nor allow description of security solutions such as the ones discussed in this paper. The security extensions to FAML described in this paper will allow software developers to describe security solutions and produce secured work products as the system is developed.

Our extensions maintain FAML's methodology-independence in the sense that it can be used to document work products (e.g. designs) created from any one of the "branded" agent-oriented methodologies, such as Prometheus [29], or those created as part of a situational method engineered from method fragments, as in the agent extensions to the OPEN Process Framework (see e.g. [11]). In both cases, the work products documented using the proposed FAML extensions for security are amenable to act as input to a method-driven engineering lifecycle from which, after appropriate rule-based transformations, code can be generated [30].

In extending our work with FAML to represent secured MAS work products, we preserve the original structure of a 2x2 matrix (Fig. 1) covering the various perspectives of the work products involved in developing a MAS. In our security-enhanced framework, managing the MAS-specific security requirements is decentralized and is relegated to the individual agents forming the system. In theory, all the security requirements of a multi-agent system fall into one of two categories, i.e. either system-wide security requirements (SSRs) or agent-specific security requirements (ASSRs). Therefore, although the metamodel does not provide specific units corresponding to specific security requirements/solutions, such as authorisation or authentication, it does provide developers with the basic units that can be used to model even complex security requirements/solutions. The definition of a metamodel that provides specific units for security requirements/solutions would be almost impossible and certainly impractical – impossible since an inevitable limitation would be to prove in a concrete way that the abstracted notions in the metamodel are sufficient for modelling all possible security requirements arising from MAS applications (similar to the impossibility of proving that an information system is

100% secure); and impractical since it would not allow adaptation to different methodologies, thus restricting the developer. Our security modelling framework is decentralised and more general than mediated security solutions such as [31]. It is also different from the works of [12, 16, 18, 21, 22], discussed in Section 2, in that we focus on the simultaneous treatment of three important properties of a software agent: autonomy, mobility and co-operation. Although these properties are important for many agent systems, particularly those in situations where security is paramount, the above approaches do not consider all of them. For instance, the work by Liu et al., and Mouratidis et al., fail to consider mobility whereas Nemo neglects mobility and autonomy.

Work more related to our suggested approach is that of Mouratidis *et al.* [24] on the definition of an architectural description language (ADL) to specify secure multi-agent systems. In that work, a set of design primitives is proposed and conceptualized using the Z specification language to capture a "core" architectural model to build secure MAS architectures. However, there are two important differences with our approach. First of all, the approach of [24] does not consider mobility and, secondly, as stated by the authors of that work, it lacks a suitable set of core abstractions, inspired by organizational metaphors, to be used during the design of the secure multi-agent system architecture. Therefore, we believe that our work complements that work by providing that missing set of core abstractions. Finally, our work is also distinct in that it is methodology-independent in the same way that in the object-oriented world, the use of UML [28] for documenting work products is not restricted to any specific methodology. It can be used to enhance the work products of any MAS methodology. It is a necessary step towards using Model Based Security for multi-agent system development.

On the other hand, our work is not complete and there are still a few outstanding questions that are raised by the introduction of additional agent-related attributes such as ownership, e.g. can agents themselves be *owned?,* and advanced agent mobility, such as dynamic routing. Further work is required to complete our set of security modelling units taking into account these additional agent attributes.

We have begun an initial verification of this security-enhanced metamodel of FAML by applying it to the analysis of security requirements of a community-based peer-to-peer web search engine. This verification will be developed further to fulfil the highest levels of complexity in security requirements taking into account roaming agents and their dynamic routing requirements. This further verification will likely overlap with additional development of the security-enhanced metamodel described here. Our future work will also link the development of secure MASs and their related access policies with risk management standards (e.g. ISO17799, ISO7498/2) as applied at an organizational level. This would guide MAS developers in making inevitable trade-off decisions of security versus cost and functionality. Authentication, intrusion detection and recovery require more computation and storage of relevant features of the involved interactions and resources by each agent. Diverting too many resources (e.g. storage, computation) towards security may indirectly limit the functionality of the system. Investing in reducing security has a point of diminishing returns, most of the benefits being reaped with the first chunk of the cost. When a MAS and its agents are modelled with our decentralised security features, it will be possible for security managers, using appropriate security policies, to achieve an optimum balance between functionality and security and to ensure that such systems are capable of protecting themselves and capable of authenticating both their incoming and outgoing interactions. In future work, we will guide developers in tackling such complex trade-off decisions.

References

[1] G. Beydoun, C. Gonzalez-Perez, B. Henderson-Sellers, G. C. Low, Developing and Evaluating a Generic Metamodel for MAS Work Products, in: A Garcia, R Choren, C Lucena, P Giorgini, T Holvoet, A Romanovsky (Eds) Software Engineering for Multi-Agent Systems IV: Research Issues and Practical Applications. Springer-Verlag, Berlin, 2006, pp 126-142.

[2] G. Beydoun, G. Low, H. Mouratidis, B. Henderson-Sellers, Modelling MAS-Specific Security Features, Proceedings of the Workshop on Exploring Modeling Methods for Systems Analysis and Design EMMSAD'07), Trondheim, Norway, Tapir Academic Press, 2007, pp. 179-188.

[3] G. Beydoun, N. Tran, G. Low, B. Henderson-Sellers, Foundations of Ontology-Based Methodologies for Multi-agent Systems, in:Springer LNCS, 2006, pp 111-123.

[4] N. Borselius, Security in Multi-Agent Systems, International Conference on Security and Management (SAM02), Las Vegas, Nevada, USA, CSREA Press, 2002, pp. 31-36.

[5] M. Bradshaw, Software Agents, MIT Press, Cambridge, MA, 1997.

[6] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, A. Perini, TROPOS: An Agent-Oriented Software Development Methodology, Journal of Autonomous Agents and Multi-Agent Systems 8(3) (2004) 203 - 236.

[7] S. Brinkkemper, Method Engineering: Engineering of Information Systems Development Methods and Tools, Information and Software Technology 38(4) (1996) 275-280.

[8] M. Esteva, Electronic Institutions: From Specification To Development, Artificial Intelligence Research Insitute, UAB - Universitat Autonòma de Barcelona, 2003.

[9] M. Esteva, D. de la Cruz, C. Sierra, ISLANDER: an electronic institutions editor, International Conference on Autonomous Agents & Multiagent Systems (AAMAS02), Italy, ACM, 2002, pp. 1045-1052.

[10]P. Giorgini, F. Massacci, J. Mylopoulos, N. Zannone, Requirements Engineering meets Trust Management, Second International Conference on Trust Management (iTrust 2004), 2004.

[11]B. Henderson-Sellers, Creating a comprehensive agent-oriented methodology - using method engineering and the OPEN metamodel, in: B Henderson-Sellers, P Giorgini (Eds) Agent-Oriented Methodologies. Idea Group, 2005, pp 368-397.

[12]M.-P. Huget, Nemo: An Agent-Oriented Software Engineering Methodology, OOPSLA Workshop on Agent-Oriented Methodologies, Seattle, USA, 2002.

[13]C. Iglesias, M. Carijo, J. Gonzales, A survey of agent-oriented methodologies, in: Intelligent Agents IV. 1999, pp 317-330.

[14]W. Jansen, T. Karygiannis, Mobile Agent Security, National Institute of Standards and Technology, Special Publication 800-19, 1999.

[15]N. R. Jennings, An agent-based approach for building complex software systems, Communications of the ACM 44(4) (2001) 35-41.

[16]J. Jürjens, Sound methods and effective tools for model-based security engineering with UML, ICSE 2005, ACM, 2005, pp. 322-331.

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

[18]L. Liu, E. Yu, J. Mylopoulos., Analyzing Security Requirements as Relationships Among Strategic Actors, 2nd Symposium on Requirements Engineering for Information Security (SREIS'02), Raleigh, North Carolina, 2002.

[19]G. McGraw, Software Security, Addison-Wesley, Indiana, 2006.

[20]T. Mine, D. Matsuno, A. Kogo, M. Amamiya, Design and Implementation of Agent Community Based Peer-to-Peer Information Retrieval Method, Cooperative Information Agents (CIA2004), Germany, Springer-Verlag, 2004, pp. 31-46.

[21]H. Mouratidis, A Security Oriented Approach in the Development of Multiagent Systems: Applied to the Management of Health and Social Care Needs of Older People in England, Department of Computer Science, University of Sheffield, PhD, 2004.

[22]H. Mouratidis, P. Giorgini, Integrating Security and Software Engineering: Advances and Future Vision, IDEA Group Publishing, 2006.

[23]H. Mouratidis, P. Giorgini, G. Manson, Modelling Secure Multiagent Systems, AAMAS'03, Melbourne, Australia, ACM Press, 2003, pp. 859-866.

[24]H. Mouratidis, M. Kolp, S. Faulkner, P. Giorgini, A Secure Architectural Description Language for Agent Systems, 4th International Joint Conference on Autonomous Agents and MultiAgent Systems, Utrecht- the Netherlands, ACM Press, 2005, pp. 578-585.

[25]J. Odell, H. Van Dyke Parunak, B. Bauer, Extending UML for agents, Agent-Oriented Information Systems Workshop, 17th National Conference on Artificial Intelligence, Austin, TX, USA, 2000, pp. 3-17.

[26]OMG, MDA Guide Version 1.0.1, OMG document omg/03-06-01, 2003.

[27]OMG, OMG: Unified Modeling Language Superstructure, Version 2.0, OMG document formal/05-07-04, 2005.

[28]OMG, Unified Modelling Language Specification, version 1.4, Object Management Group, formal/01-09-68 through 80 (13 documents), 2001.

[29]L. Padgham, M. Winikoff, Developing Intelligent Agent Systems. A Practical Guide, J. Wiley & Sons, Chichester, 2005.

[30]O. Pastor, J.-C. Molina, Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling, Springer, Berlin, 2007.

[31]J. A. Rodriguez, On The Design and Construction of Agent-Mediated Electronic Institutions, Artificial Intelligence Research Insitute, UAB - Universitat Autonòma de Barcelona, 2003.

[32]V. Roth, Programming Satan's agents, International Workshop on Secure Mobile Multi-Agent Systems, Montreal, Canada, 2001.

[33]V. T. D. Silva, C. J. P. D. Lucena, From a Conceptual Framework for Agents and Objects to a Multi-Agent System Modeling Language, Autonomous Agents and Multi-Agent Systems 9(1-2) (2004) 145-189.

[34]Q. N. N. Tran, G. Beydoun, G. C. Low, C. Gonzalez-Perez, Design of a Peer-to-Peer Information Sharing MAS Using MOBMAS (Ontology-Centric Agent Oriented Methodology), AOIS@CAiSE 2006, Luxembourg, 2006, pp. 52-63.

[35]M. Wooldridge, An Introduction to Multi Agent Systems, Wiley, Chichester, 2002.

[36]M. Wooldridge, P. Ciancarini, Agent-Oriented Software Engineering: The State of the Art, in: P Ciancarini, M Wooldridge (Eds) Agent-Oriented Software Engineering. Springer-Verlag, 2001, pp 1-28.

[37]B. Yee, Monotonicity and Partial Results Protection for Mobile Agents, 23rd International Conference on Distributed Computing Systems, Rhode Island, 2003.

Figures

Fig. 1. The 2x2 matrix that is used to define four typical FAML metalevel diagrams

Fig. 2. System-related design-time classes. [The diamond notation indicates a generic whole-part relationship] (after [1])

Fig. 3. Run-time, environment-related classes (after [1])

Fig. 4. Agent-internals design-time classes (after [1])

Fig. 5. Agent-internals run-time classes (after [1])

Fig. 6. System-level (agent-external, design-time) classes. (Note that the duplication of the Role class is only to simplify the layout) (updated from [2])

Fig. 7. Environment-level (agent-external, runtime) classes (updated from [2])

Fig. 8. Agent definition-level (agent-internal, design-time) classes (updated from [2])

Fig. 9. Agent-level classes (agent-internal, runtime) (updated from [2])

	Agent - external	Agent - internal
Design time	System level	Agent Definition level
Run time	Environment level	Agent level

Fig. 1. The 2x2 matrix that is used to define four typical FAML metalevel diagrams

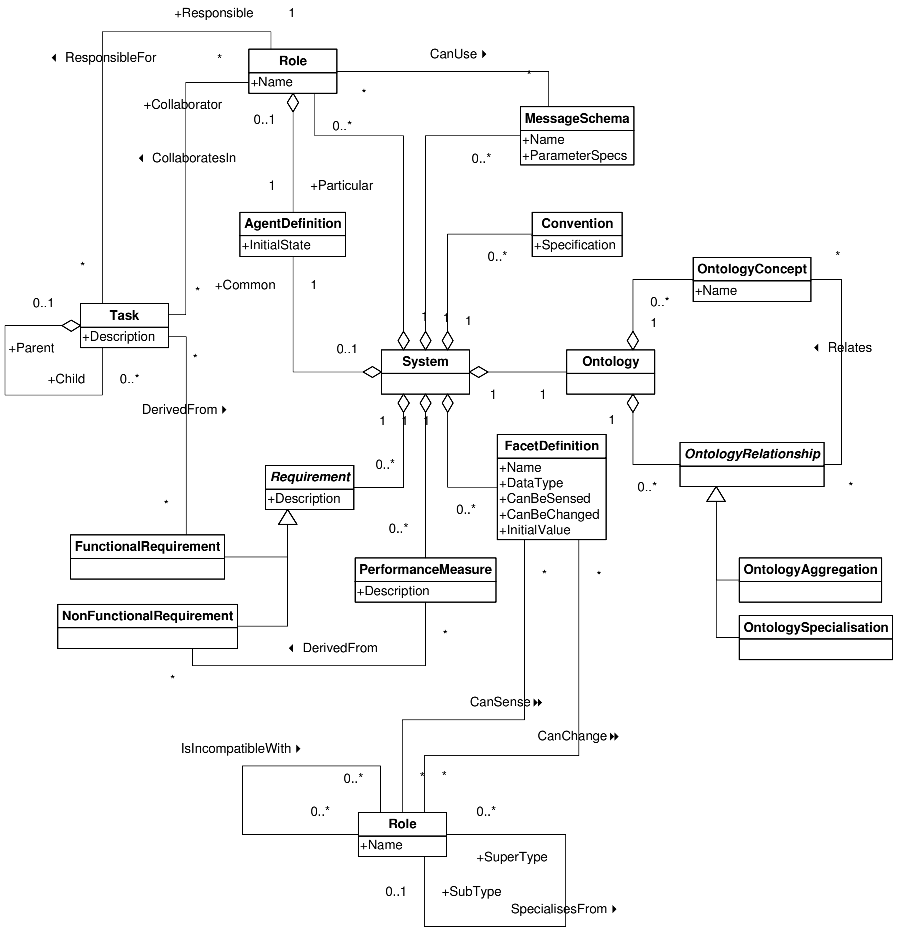

Fig. 2. FAML original System-related design-time classes. The diamond notation indicates a generic whole-part relationship (after [1]). The concept *Role* is repeated for layout convenience.

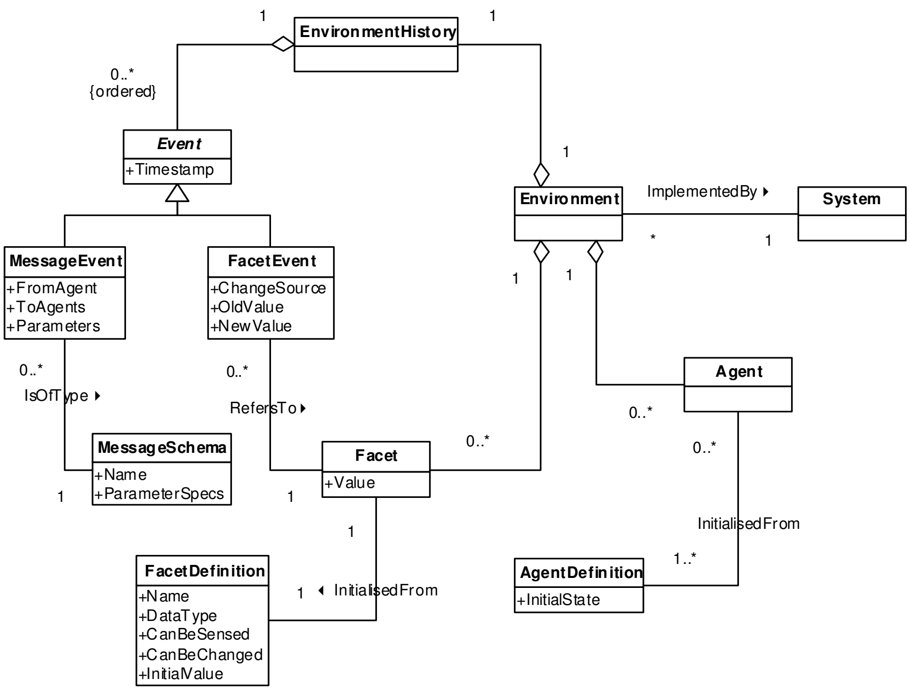

Fig. 3. FAML original run-time, environment-related classes (after [1])

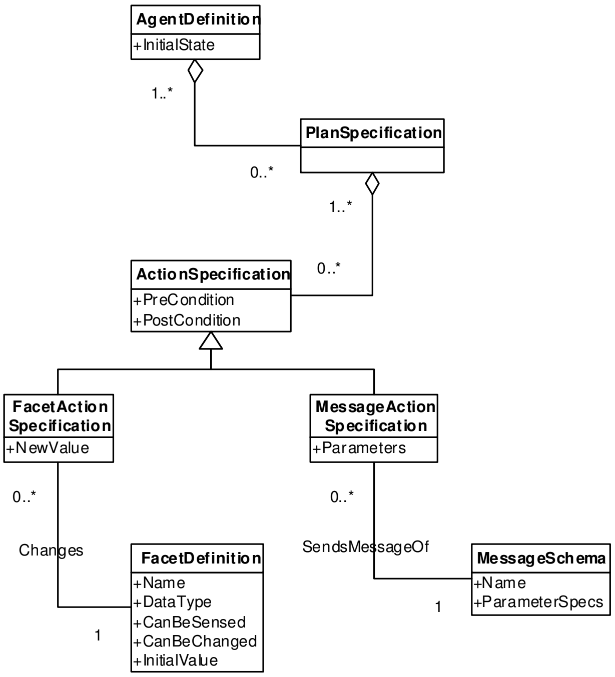

Fig. 4. FAML original agent-internals design-time classes (after [1])

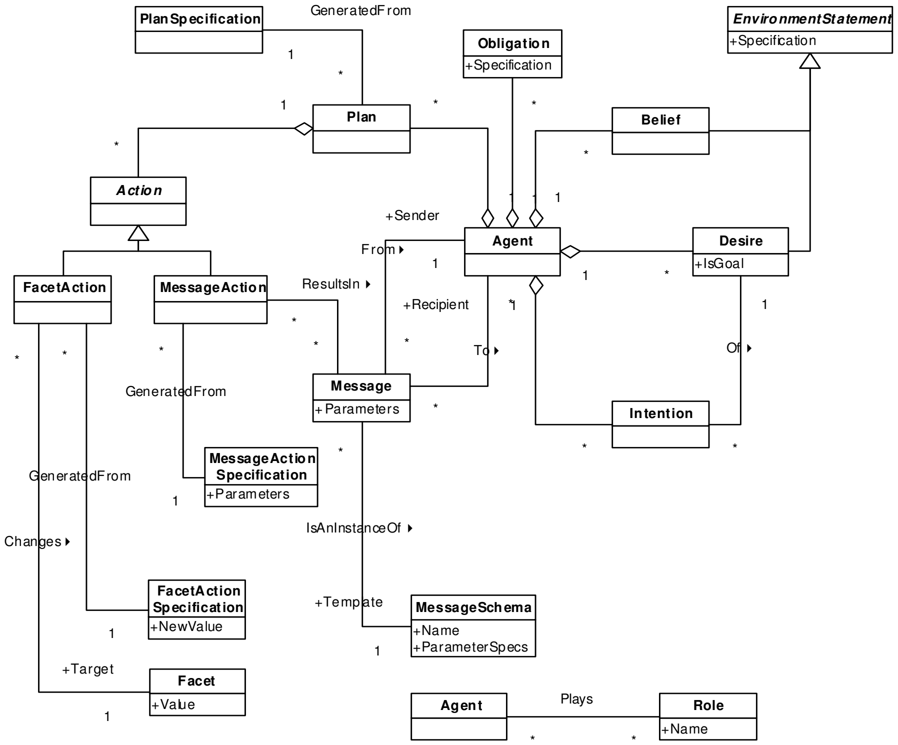

Fig. 5. FAML original agent-internals run-time classes (after [1]).

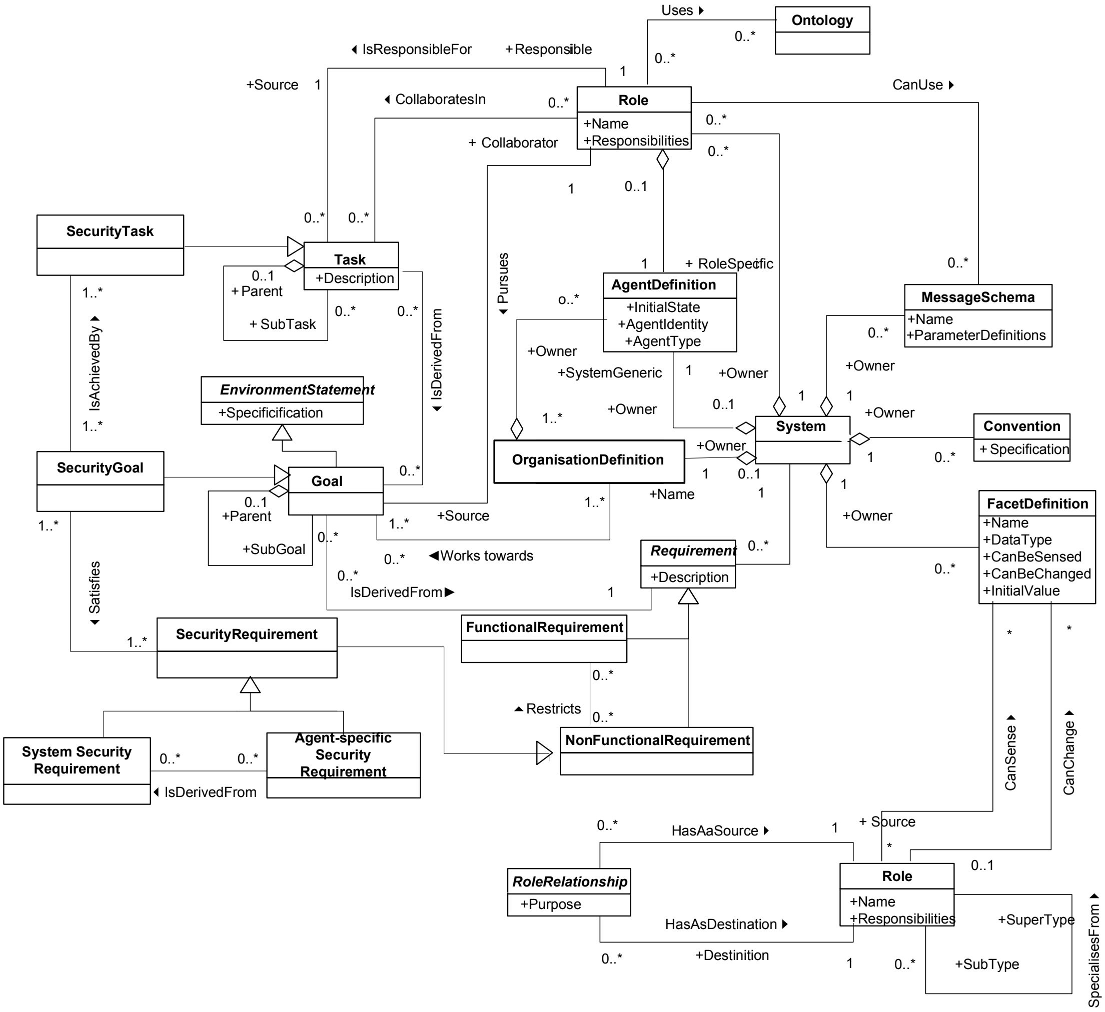

Fig. 6. System-level (agent-external, design-time) classes: (updated from [2]) This diagram shows the addition of (System) Goals and Organisation Definition (cf.h Fig. 2). More importantly, it shows the inclusion of *Security Requirement* and its refinement into corresponding security goals and tasks.

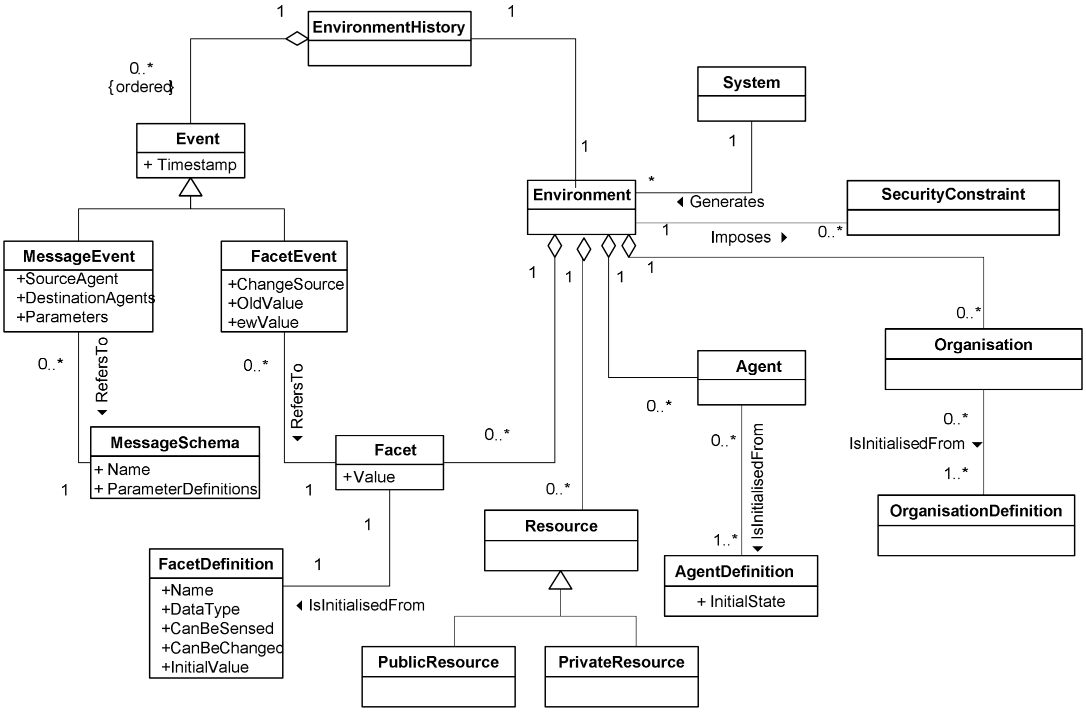

Fig. 7. Environment-level (agent-external, runtime) classes (updated from [2]): This diagram shows the global security concerns at runtime, including the resources involved and the security constraints.

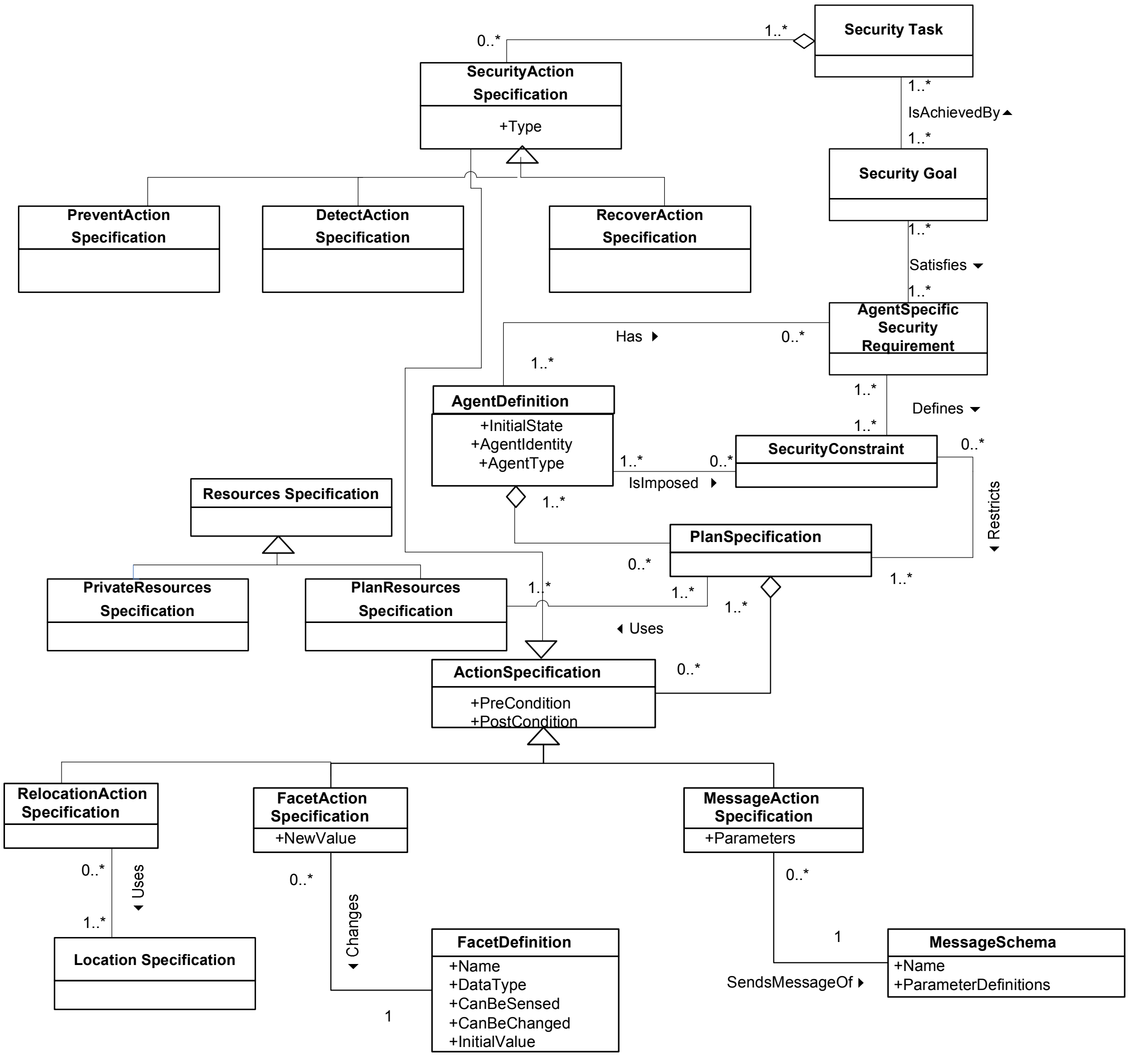

Fig. 8. Agent definition-level (agent-internal, design-time) classes (updated from [2]): This diagram shows how system security tasks are refined into various types of specific security actions associated within individual agents of the system.

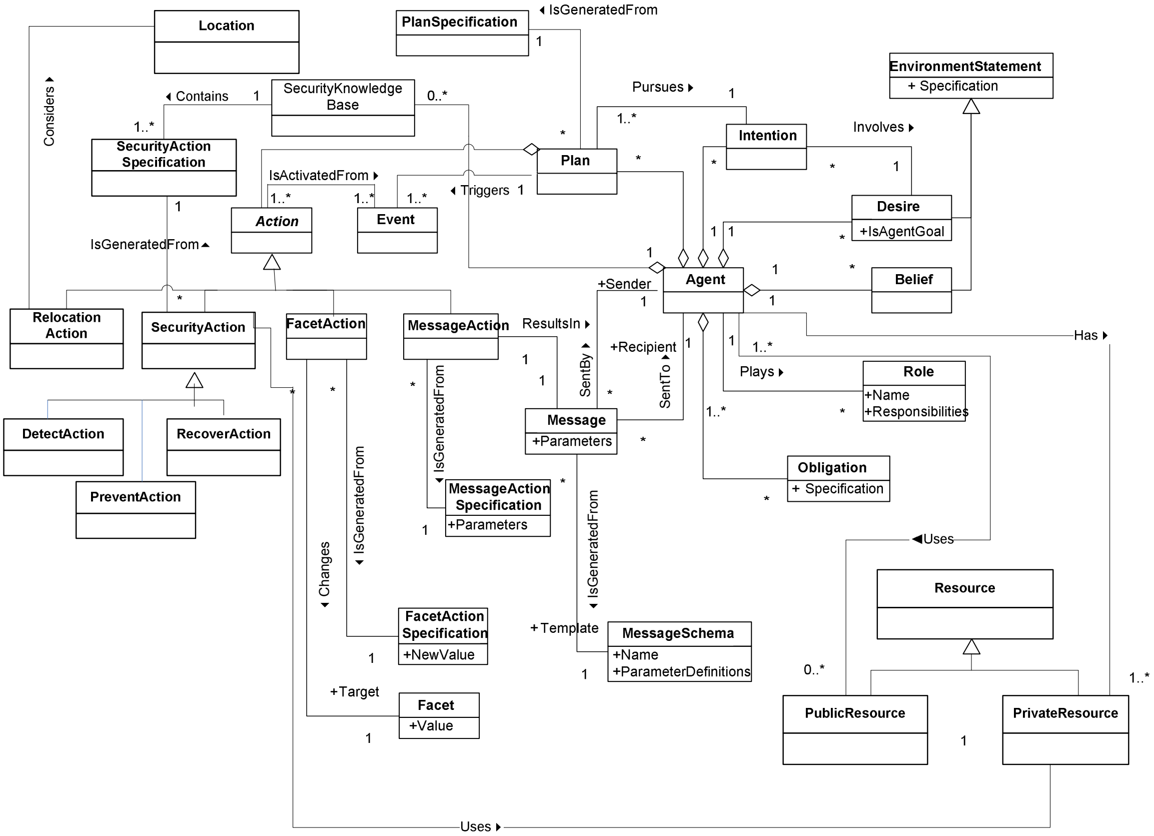

Fig. 9. Agent-level classes (agent-internal, runtime) (updated from [2]). Agents action specification (from Figure 8) generate the actual actions at run-time. These include security actions which are activated due to run-time events.

Tables

Table 1. Design-time concepts and their definitions (after [1])

Term	**Definition**
Action Specification	*Specification of an action, including any preconditions and postconditions.*
Agent Definition	Specification of the initial state of an agent just after it is created.
Convention	Rule that specifies an arrangement of events expected to occur in a given environment.
Environment Statement	A statement about the environment.
Facet Action Specification	Specification of a facet action in terms of the facet definition it will change and the new value it will write to the facet.
Facet Definition	Specification of the structure of a given facet, including its name, data type and access mode.
Functional Requirement	Requirement that provides added value to the users of the system.
Message Action Specification	Specification of a message action in terms of the message schema and parameters to use.
Message Schema	Specification of the structure and semantics of a given kind of messages that can occur within the system.
Non-Functional Requirement	Requirement about any limits, constraints or impositions on the system to be built.
Ontology	Structural model of the application domain of a given system.
Ontology Aggregation	Whole/part relationship between two ontology concepts.
Ontology Concept	Concept included in a given ontology.
Ontology Relationship	Relationship between ontology concepts.
Ontology Specialisation	Supertype/subtype relationship between two or more ontology concepts.
Performance Measure	Mechanism to measure how successful the system is at any point in time.
Plan Specification	An organised collection of action specifications.
Requirement	Feature that a system must implement.
Role	Specification of a behavioural pattern expected from some agents in a given system.
System	Final product of a software development project.
Task	Specification of a piece of behaviour that the system can perform.

Table 2. Run-time concepts and their definitions(after [1])

Term	Definition
Action	Fundamental unit of agent behaviour.
Agent	A highly autonomous, situated, directed and rational entity.
Belief	An environment statement held by an agent and deemed as true in a certain timeframe.
Desire	An environment statement held by an agent, which represents a state deemed as good in a certain timeframe.
Environment	The world in which an agent is situated.
Environment History	The sequence of events that have occurred between the environment start-up and the present instant.
Environment Statement	A statement about the environment.
Event	Occurrence of something that changes the environment history.
Facet	Scalar property of the environment that is expected by the agents contained in it.
Facet Action	Action that results in the change of a given facet.
Facet Event	Event that happens when the value of a facet changes.
Goal	Ultimate desire.
Intention	A committed desire.
Message	Unit of communication between agents, which conforms to a specific message schema.
Message Action	Action that results in a message being sent.
Message Event	Event that happens when a message is sent.
Obligation	Behaviour expected from an agent at some future time.
Plan	An organised collection of actions.

Table 3. Context Complexity of MAS

Vulnerability level	**Description of Context Complexity Level**
0	A single agent system on a single machine.
1	MAS running on a single machine– controlling access to agents' resources during co-operation
2	Distributed MAS (same as MAS if communication is secured)
3	Weakly mobile agents
4	Strongly mobile agents

Table 4. The set of identified security modelling units

Security Term	**Definition**
Agent Specific Security Requirement (design-time)	A security requirement that is true only for a specific agent of the system. This is a subclass of Security Requirement.
DetectAction (run-time)	A detect action is an action that results in an agent initiating a detection procedure aiming to detect potential security breaches. This is a subclass of SecurityAction.
DetectAction Specification (design-time)	A detect action specification is a specification of an action that an agent can take to detect a possible security incident, such as a security attack. This is a subclass of SecurityActionSpecification.
PreventAction (run-time)	A prevent action is an action that results in an agent initiating a prevention procedure aiming to prevent potential security breaches. This is a subclass of SecurityAction.
PreventAction Specification (design-time)	A prevent action specification is a specification of an action that an agent can take to prevent a security incident, such as a security attack. This is a subclass of SecurityActionSpecification.
RecoverAction (run-time)	A recover action is an action that results in an agent initiating a recovery procedure after a security incident. This is a ubclass of SecurityAction.
RecoverAction Specification (design-time)	A recover action specification is a specification of an action that an agent can take to recover from a security-related incident. This is a subclass of SecurityActionSpecification.
Security Action (run-time)	A security action is an action that results in a security-related action been taken. This is a subclass of Action.
Security Action Specification (design-time)	A security action specification is a specification of the security action in terms of the security action type to use. This is a subclass of ActionSpecification.
Security Constraint (design-time)	A security constraint is a statement (expressed in a logical or informal way) used to precisely define the restrictions imposed on an agent due to security requirements.
Security Goal (design-time)	A security goal is a specification of a security-related state that the system/agent tries to achieve. This is a sub-class of SystemGoal.
Security KnowledgeBase (run-time)	A security knowledge base represents the security knowledge that an agent needs to be able to perform security-related actions.
Security Requirement (design-time)	A security requirement is a desirable security-related requirement that the system/agent must demonstrate. This is a subclass of Non-Functional Requirement.
Security Task (design-time)	A security task is a specification of a piece of security behaviour that a system and/or an agent can perform. This is a subclass of SystemTask.
System Security Requirement (design-time)	A system security requirement is a security requirement that is true for all the agents of the multi-agent system. This is a subclass of Security Requirement.

Table 5. FAML modified or new concepts, accommodating new security concepts (shown in Table 4). Old concepts that are modified are shown in italics, the rest are new concepts.

Non-Security Term Modified	**Definition**
AgentDefinition (design-time)	An agent specification is a specification of the initial state of an agent just after it is created. It has three attributes: InitialState, AgentIdentity and AgentType (e.g. mobile versus static)
Goal (design-time)	A goal is a specification of a state of the environment that the system tries to achieve. This is a subclass of EnvironmentStatement.
Location (run-time)	Location provides information about places where an agent can reside within the system.
LocationSpecification (design-time)	Location specification is information about places where an agent can reside within the system.
NonFunctionalRequirement (design-time)	A non-functional requirement is a requirement about any limits, constraints or impositions on the system to be built. This is a subclass of Requirement.
ResourceSpecification (design-time)	A resource specification specifies something that has a name, may have reasonable representations and that can owned. The ownership of a resource is connected with the right to set policy on the resource.
PlanReseourceSpecification (design-time)	This is a specification of resources that are used in the Plan Specification.
PrivateResourceSpecification (design-time)	This is a specification of those resources that are only visible and available to the individual agent.
RelocateAction (run-time)	An agent can relocate to another location based on information contained in the interaction log and agent-location if an interaction gives access to resources when it shouldn't.
RelocationActionSpecification (design-time)	Relocation action specification is a specification of how an agent can move to another location based on information contained in the interaction log and agent-location if an interaction gives access to resources when it shouldn't.