The ShaMAN Agent Meta-model

Steve Goscchnick, Sandrine Balbo and Liz Sonenberg
Department of Information Systems, University of Melbourne, Australia, 3010
{stevenhg, Sandrine, lsonenberg}@unimelb.edu.au

Abstract. In this paper, we detail recent research on agent meta-models. In particular, we introduce a new agent meta-model called ShaMAN, created with a specific focus on computer game development using agent systems – an application domain that is fertile ground for Agent-oriented concepts, methodologies and tools. ShaMAN was derived by applying the concept of Normalisation from Information Analysis, against a super-set of agent meta-model concepts from the meta-models investigated.

Keywords: Agent-oriented, Agent architecture, Multi-Agent Systems, Meta-model, Agent Meta-models, Computer game development, Agents in computer games

1 Introduction

Agent-oriented (AO) architectures and methodologies are the main interest of the research outlined here, with a focus on the application domain of computer games. While the games domain is one of many complex application areas that are prime territory for exploitation of AO techniques, tools and architectures, we are specifically interested in extending current AO concepts to further facilitate game specification and development. A consequence of this study is the identification of possible generic features to add to an AO meta-model.

1.1 Motivation

Computer games invariably have a graphic user interface (GUI) whether they are on PCs, dedicated game consoles, or mobile phones. Additionally, many games are multi-user, over either a proprietary network or increasingly the Internet, and as such some data is often shared between multiple simultaneous users. Neither graphic interfaces and their associated event models, nor the sharing of data resources is well considered in the current AO architectures and frameworks, but computer games make heavy use of all three.

Furthermore, in the more recent worlds of Service Oriented Architecture (SOA) and Web 2.0, the strength of the relational and transactional DBMS (database management system), has come to the fore over distributed object approaches. This is because DBMS support sharing of data concurrently and efficiently, allowing multiple users in a distributed environment to update information without technical difficulty. We felt that some concepts and techniques from the relational database field may also yield benefits for the AO paradigm.
1.2 A Gap in Agent Architectures

From the start AO has been socially-oriented such that inter-agent communication – a form of event - is typically allowed for with an Agent Communication Language (ACL). However, the AO paradigm has followed the initial OO programming languages, in not doing anything within the architecture or the constructs of the languages themselves, with regard to the GUI interface or non-agent event-handling. Similarly, in many AO architectures, frameworks and methodologies, data and informational resources are usually left to non-agent modelling and implementation.

The gaps addressed in our research, is to achieve an AO meta-model that engages with the typical computer that people currently use, at the level of a GUI metaphor that gets rendered down to the pixel level, with events down to the keyboard and mouse click level, and one that addresses shared data and other resources that are often not Agent-oriented in their design and construction.

1.3 Meta-models

Much of the research discussed here is centred around agent meta-models expressed in either: UML class diagram notation, and Entity Relation (ER) notation [2], which requires a little explanation up-front.

Meta-models expressed in UML class diagrams are now commonly used in both AO [1,12,13] and OO [17] research and development domains: to represent state-holding entities to communicate base ideas; and as a useful means to compare different agent systems or architectures [7,12]. To facilitate the process of comparison, it is useful to represent as many aspects of an architecture as possible in a single meta-model diagram.

Entity Relation Modelling. In the paradigm of Information Analysis (IA), which began before the OO paradigm, an Entity Relation (ER) diagram [2] is used to represent the conceptual data model of an application, usually then built in a relational DBMS. ER models are not too dissimilar from UML class diagrams and indeed there are several popular general-purpose contemporary technologies that allow OO developers to maintain the state of their objects in relational DBMS as a persistent state machine (Hibernate, Apache Cayenne). And furthermore, meta-models have also been used to compare information analysis methodologies and ontologies [6], in a similar endeavor to the meta-model comparisons of recent agent research.

Normalisation. A well-founded technique from Information Analysis (IA) used to improve ER models that did not crossover into the OO paradigm is the concept of Data Normalisation. In this process the ER model is put into normal form using a technique derived from relational mathematics by Codd [3]. The ER model resulting from normalising a preliminary ER model, is considered to be in a state that is ideal for future change, and one that causes the least anomalies to current operations upon it – i.e. when inserting, deleting and modifying the state held in the entities. It is usually applied in IA to a model as a quality control procedure, however, Normalisation can also be used as a bottom-up design technique enabling the analyst to methodically deduce a well-formed design, from a loosely gathered set of concepts, by applying the rules of Normalisation [15] to them.

Agent Concepts. Given that there is currently no universally accepted single meta-model for AO systems, when we first looked to agent concepts and architectures with computer games in mind, we examined the meta-models of several well-known agent architectures and methodologies (AAII [16], GAIA [21,22], Tropos [1,11], TAO/MAS-ML.
[5], Prometheus [18] and several that are less well known (ROADMAP [13,14], ShadowBoard [9,10], GoalNet [19]), to explore the commonalities and differences between them. In addition, given our identification of a gap in the AO paradigm down at the input device event level, we included in our study several well-known meta-models from the Task Modelling field, that has its roots in the interaction between human users and computational devices.

Fig. 1. The ShaMAN Agent Meta-model

At a fundamental level an agent meta-model is an entity model, thereby abiding by all of those conventions of a notation that an application model adheres to. In this research, in a
bottom-up design manner, we applied Normalisation to a superset of the agent concepts found in the agent meta-models and the task meta-models considered, and arrived at a normalised agent meta-model named ShaMAN, presented in figure 1. As a normalised model it is both: flexible to ongoing requirements upon the meta-model itself (should it need future enhancements); and as a computation framework built upon it, is a model least susceptible to anomalies arising from changes of state with respect to the existing concepts represented in it.

**Overview.** In Section 2 we introduce the ShaMAN agent meta-model proper, which resulted from the Normalization of all of the concepts from the aforementioned agent meta-models. In order to explain the entities in figure 1 efficiently, we present several groupings of the entities from the meta-model in detail, and then describe the flexibility they bring to building applications using the model. In Section 3, we compare some concepts from the ShaMAN meta-model with those other meta-models investigated. In Section 4 we describe the current application of ShaMAN to the development of a computer game called BranchOut. In Section 5 we look to future work related to ShaMAN.

2. The ShaMAN Meta-model

We arrived at the ShaMAN meta-model depicted in figure 1, by taking concepts from a number of existing agent meta-models - namely: AAI, Gaia, Roadmap, GoalNet, TAO, Tropos, ShadowBoard and Prometheus (and a some Task Meta-models not discussed here) – analyzing their similarities and differences, incorporated a few extra requirements from the computer games genre, and then normalised the resultant set of entities. The following sections describe aspects of the ShaMAN meta-model in more detail.

![Fig. 2. The Locale sub-section of the meta-model](image)

2.1 Locales for Computer Games

The domain of applications of interest to us in this research is that of computer games, which invariably interact with the player through the usage of a human-machine interface, for example a *screen of one size or another*. The Locale sub-section of the ShaMAN model lets us model the visual metaphors and the screen interaction between player/user and screen characters of a game, right in the AO model itself, rather than leaving that to some other modeling paradigm such as OO. While several of the agent meta-models investigated do have constructs for the agent *environment*, none specifically
model the computer screen as the primary representation of the environment to the user. In ShaMAN this screen representation of a sub-section of the agent’s environment called a Locale in homage to Fitzpatrick’s [8] definition of a Locale as a generalised abstract representation where members of a Social World [20] inhabit and interact. Figure 2 represents that sub-section of the ShaMAN meta-model that represents Locales within games. Note: the simplified crow’s-foot (zero, one or many) ER notation is used in these detailed figures of sub-sections of the meta-model, to increase the readability.

The Locale entity may have sub-locales allowing hierarchies of Locales. Locales are a generic concept representing some spatial construct presentable on the screen, e.g. room, outdoor area, section of a board-game, etc. By way of describing the entities in figure 2, we will refer to the game screen in figure 3 below as a concrete example of Locale.

The screen depicts the bedroom of the player’s character within the game BranchOut, and that room is represented as a Locale in ShaMAN. The HotSpot entity represents any area on the screen that is interactive, in the sense that whenever the user either clicks or passes over that area on the screen (or if a HotSpot has the focus, from a keystroke point-of-view), certain interaction between the user and the game may take place – for example clicking on the digital clock on the bedside table open a window that displays the current game-time. HotSpots are a generic concept for such online screens, whether the game is presenting a 2D, a 3D or some abstract representation, the interaction with a standard display is 2D and involves area. HotSpot has two relationships with Locale, one named to and the other named from – simply enabling navigation between Locales via HotSpots.

![Fig. 3. Depiction of a Locale in a Computer Game being developed using ShaMAN](image)

A HotSpot may also link to an OnSiteResource entity. These are Resources that live in the Resource entity (which may involve a hierarchy of Resources). Resources are typically programmed entities that are not Agents in their conception nor development (but Resource may also represent real objects in the real world). The digital clock example is a fully-functioning clock object, which, via clicking on it displays a fuller interface to the digital clock Resource. OnSiteResource is an associate entity – a representation which allows the same Resource to be used in multiple Locales, e.g. a clock in many rooms drawing upon the same programmed code. A HotSpot may also have a relationship with the entity LinkCondition, which in turn links to a Goal via a relationship called has-turdle. This allows the game developer to enforce conditions to be met: e.g. before the player may advance to another Locale, or before they may use a particular Resource.

Locale is also directly linked to two other related entities in Attendee and Inhabitant. Attendee is an associative entity that records all occupants in a particular Locale over time,
retaining a record of when agents (or human avatars) entered the Locale and when they left it, if they are no longer present. It is linked to the agent’s Role during that occupation via AgentRole, and also to the SocialWorld they were engaged in when they did so. This historical aspect of the Attendee is useful in providing and/or recording a back-story for any particular agent-oriented game character – a necessary aspect of realistic game creation. In contrast to Attendee, the Inhabitant entity represents the current occupants of the Locale. It has a direct link to the Agent entity, and is used when the overhead of SocialWorlds and agent Roles aren’t a concern.

![Diagram of ShaMAN message flow]

Fig. 4. Message flow within ShaMAN

2.2 Communication via SpeechFlow

Some computer games have large numbers of agents, and very often these agents are categorized by the roles they serve. When it comes to communication between agents, there is the need for communication at several levels: one-to-one; one-to-a-group of agents in a particular Social World; one-to-all agents filling a specific role (e.g. Captain). Note: Human players (human-in-the-loop) are considered to be agents, from a communication point of view. The SpeechFlow sub-section of the ShaMAN meta-model as portrayed in figure 4 addresses these three required levels of communication in gameplay.

The SpeechFlow entity is at the heart of all interaction within ShaMAN. An agent communicates to other agents or to humans-in-the-loop by generating an instance of SpeechFlow. It does so while acting in some Role and while working upon some Goal associated with that Role. The entity that represents the instance of those two things for a particular agent is AgentRoleGoal. While the AgentRoleGoal instance may generate many SpeechFlow instances, each one of them is always links to just one ActionType, which are in turn determined by a particular agent communication language (ACL).

While a particular instance of AgentRoleGoal is the source of any given instance of SpeechFlow it can have one (or more) of three possible receivers: AgentRole is a given individual recipient agent while acting in a particular role; SocialWorld means that the message will be broadcast to the whole membership of a social world how ever many members it has (via Member – see figures 1 or 2); or, SocialRole which is a particular role common to many SocialWorlds, such as ‘Treasurer’, or, a game example such as ‘Captain’ – E.g. all ‘Captains’ of SocialWorld’s of type ‘Ship’. The three receives relationships are all zero-or-one to many, because they may be either alternative or parallel receivers of a specific message.

2.3 The Goals, Roles, Responsibilities and Tasks of Agents

Computer games often have the need for intelligent, intentional, proactive and autonomous game characters that interact both with the human players and with other characters in a game. These properties are the harbingers of AO systems, and the sub-group of
entities from ShaMAN meta-model in figure 5, represent the entities that appear most frequently (but not consistently) in one form or another, in many of the agent meta-models that we examined.

Referring to figure 5 there are four entities in this sub-model of ShaMAN that have hierarchies of sub-elements of the same type, namely: Goal, Role, Agent and Task. The associate entity between Goal and Role called Responsibility represents the responsibilities of a particular Role. A given Responsibility instance is fulfilled via an instance in the AgentRoleGoal entity, by being enacted or performed by an Agent that takes on that Role. An Agent may have many Roles and the AgentRole entity represents this multiplicity.

Traditionally in task modeling, a hierarchy of related tasks is performed to achieve a goal that sits at the head (or root) of the task hierarchy. In ShaMAN, the AgentRoleGoal entity represents such a root Goal, while such a task hierarchy is represented Task together with the self-relationship (unary relationship) called sub-task.

The completion of a Task may spark a SpeechAct. SpeechActs (zero, one or many of them) may also be generated directly by the AgentRoleGoal entity. Note that SpeechAct is related to the ActionType and SpeechFlow entities described earlier in section 2.2.

**Fig. 5. Goals, Roles and Tasks in ShaMAN**

Goals within ShaMAN are expressed as GoalX(term1, term2, … termN), like a predicate in the predicate logic sense. (I.e. by setting one or more of the terms as a constant, and leaving one or more of the terms as variables, a logic language such as Prolog will accept such a predicate as a goal, and will set about solving it). Even though this predicate format for goals conforms with logic languages, it is also an acceptable way to specify methods capable of achieving goals in imperative languages. Likewise, it is an equally acceptable format for expressing RPC-oriented (Remote Procedure Call) web services, and database queries in a Query-by-Example format. Further, it is an acceptable way to express a goal via an anti-tuple in Tuplespace systems.

In ShaMAN a Term can be a simple variable (or a literal) but it may also include constraints, e.g. while the variable Temperature is an acceptable term within a ShaMAN goal, so is the constraint: during(12 < Temperature < 100). A term expressed as such is an invariant constraint, meaning that it remains in force for the life of the goal to which it is associated. A second type of constraint can be expressed in a ShaMAN term as follows: before(12 < Temperature < 100) – meaning that the constraint must hold before the goal can be begin to be solved. An example of a third form of constraint definition is after(Temperature = 100), representing a constraint that exists following a successful completion of the goal in question.
Goals will often have sub-goals in a hierarchy of goals to be achieved. One such sub-goal will be associated with a matching sub-role, assigned to an agent via an instance of the AgentRole entity. During execution of a ShaMAN application, sub-agents can be fired in a downward direction via the need to achieve the sub-goals of parent goals - termed goal-driven execution - or, they can be fired from below, where a SpeechAct has been sent from further down the sub-agent chain, and the upper level goal has to be solved or rerun - termed data-driven execution. Data-driven execution often eventuates when a sub-agents retrieves new information from an external service such as a web service, or from another agent across agent hierarchies or across Social worlds.

2.4 Social Worlds in ShaMAN

Individual Agents can be members of one or more SocialWorlds. Their membership begins with an instance in the Member entity. Agents are related to the Member entity via the one-to-many relationship involved-as between their entry in the AgentRole entity and Member, while SocialWorlds include multiple agents represented in Member via the one-to-many relationship includes.

![Diagram](image)

**Fig. 6. SocialWorlds within ShaMAN**

Clearly, Agents can fill multiple Roles via the AgentRole entity. The roles that are available within a particular SocialWorld are listed as instances of the SocialRole entity, which sits between Role and SocialWorld. SocialRole is a useful entity in a number of ways: it can be used to specify all the roles that will make up a SocialWorld before any Agents become members in the design phase; and, as we have already seen in section 2.2 above, at execution time SpeechFlow messages can simply be broadcast to all agents in a SocialWorld that occupy a particular SocialRole such as Captain or Treasurer.

2.5 Knowledge Tree and Resources

The Knowledge Tree part of ShaMAN consists of a hierarchy of concepts in the form of an Ontology, but which is then related to lists of resources (via the ResourceList entity) at each level in the ontology. In the functioning system based on ShaMAN the Ontology is represented as a file directory structure, and the resource list includes a multitude of file types, including images and video files used in a game, through to executable Java objects, such as the Digital Clock and the Calendar displayed in figure 3 above and discussed in section 2.1. It differs from a conventional file directory structure in that any resource can appear in multiple ResourceLists via the appear-in relationship between them.
Fig. 7. The ShaMAN Knowledge Tree

The Resource hierarchy is used to store any group of objects or resources that are naturally composed in a hierarchical form. For example, a computer consists of motherboard, hard drive/s, memory cards, etc. – which could be logically represented in the Resource tree of ShaMAN. Similarly, a user interface screen is most often a hierarchy of on-screen components – e.g. the buttons within the button pad along the bottom of the screen in Figure 3, are a part of the resource hierarchy used by a Locale that represents that screen, and linked by the OnSiteResource entity depicted at the bottom of Figure 2.

Just as a Locale can use Resource so too can an agent. It may do so through Agent-Role which is related to Resource via the associative entity AgentResource depicted at the top of Figure 7.

3. A Comparison of Agent Meta-Model Concepts

The original set of agent concepts from the agent architectures and methodologies that we used as input into the meta-model normalisation process to derive ShaMAN, are represented in a comparative format in Table 1.

While a particular comparison (e.g. ShaMAN’s Goal(tree) and Tropos’s Soft Goal/Hard Goal) approximately equates the concepts, this is a gross simplification for presentation purposes only. Sometimes the comparison is close in meaning, other times it is close in name but distant in meaning, and sometimes there is wide variance in both name and the semantics. For example the Role (tree) in ShaMAN is a hierarchy of Roles much as it is in ROADMAP, while Role in GAIA is a list of unrelated roles. ROADMAP is a good example of a meta-model that uses much of the terminology of GAIA (i.e. it began as an extension to GAIA V1), some of which have the same meaning as GAIA, some a divergent meaning, while others have no equivalence in GAIA, e.g.: ROADMAP Agents contain Services much like OO methods, and Services implement Protocols, like OO methods often implement Interfaces, but they can be dynamically bound to each other at runtime; ROADMAP Services represent both things that GAIA Services and Activities represent.

The concepts and semantics are often more divergent across meta-models that do not share as much initial theoretic grounding, or which provide a comprehensive methodology and tools. E.g. from the outset Tropos was designed to cover the full lifecycle of development covering requirements analysis, design and implementation (but not testing), while GAIA chose to only address analysis and design phases of the software development lifecycle. However, when you look beyond the concept names to the semantics, there is a good deal of overlap between most of the agent models. In ShaMAN, Goals are specified in a predicate logic sense. When they are assigned to Roles, they form Responsibilities. When they are assigned to an Agent via Responsibilities, they form AgentRoleGoals which
are similar in semantics to GAIAs Activities and Services, and to ROADMAPs Services. At runtime AgentRoleGoals are represented as Tasks, which in an implemented technology, could be instantiated as either: a web service, a logic language predicate, an imperative language/ OO method, or even an SQL Select statement. I.e. all of these technologies are capable of returning ground terms in the predicate logic sense. In the full study (currently within the first author’s PhD research program) we do examine each comparison of concept in detail, but it clearly cannot be presented in this paper for space reasons.

Table 1. ShaMAN meta-model comparison with other agent architectures and meta-models

<table>
<thead>
<tr>
<th>ShaMAN</th>
<th>KGR for RDI</th>
<th>GAIA V1</th>
<th>GAIA V2</th>
<th>RoadMap</th>
<th>Tropos</th>
<th>MAS-ML</th>
<th>DigitalFriend</th>
</tr>
</thead>
<tbody>
<tr>
<td>Act (trm)</td>
<td>Role (true), Role #</td>
<td>Role (true), Role (true)</td>
<td>Role (true)</td>
<td>Actor/Role</td>
<td>Role</td>
<td>Role (true)</td>
<td></td>
</tr>
<tr>
<td>Responsibility</td>
<td>Responsibility</td>
<td>Responsibility</td>
<td>Responsibility</td>
<td>Responsibility</td>
<td>Dependency</td>
<td>Duties, Rights</td>
<td>Responsibility</td>
</tr>
<tr>
<td>Agent (trm)</td>
<td>Agent, Control-Relationship</td>
<td>AgentType (true), AgentInstance, Service</td>
<td>AgentType (true)</td>
<td>Agent (true)</td>
<td>Agent/SoftwareAgent</td>
<td>Agent, Object</td>
<td>Agent (true)</td>
</tr>
<tr>
<td>AgentRole</td>
<td>Role</td>
<td>Role</td>
<td>(geometry)</td>
<td>Play</td>
<td>AgentRole</td>
<td>Role</td>
<td>Event</td>
</tr>
<tr>
<td>Event</td>
<td>Event</td>
<td>Event</td>
<td>Event</td>
<td>Event</td>
<td>Event</td>
<td>Event</td>
<td>Event</td>
</tr>
<tr>
<td>Access</td>
<td>Acquaintance</td>
<td>System</td>
<td>Organisation, Organisation-Pattern</td>
<td>Actor/SocialAgent</td>
<td>Organisation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SocialRole</td>
<td>System</td>
<td></td>
<td></td>
<td>Actor/Position</td>
<td>Ownership</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitor</td>
<td>Event</td>
<td>Event</td>
<td>Event</td>
<td>Event</td>
<td>Event</td>
<td>Event</td>
<td>Event</td>
</tr>
<tr>
<td>AgentRoleGoal</td>
<td>Capability, Services</td>
<td>Services, Activity</td>
<td>Activity</td>
<td>Activity</td>
<td>Activity</td>
<td>Activity</td>
<td>Activity</td>
</tr>
<tr>
<td>Task (trm)</td>
<td>Activity, Activity</td>
<td>Plan</td>
<td>Task</td>
<td>Task</td>
<td>Task</td>
<td>Task</td>
<td>Task</td>
</tr>
<tr>
<td>Event</td>
<td>Interaction</td>
<td>Activity</td>
<td>Message</td>
<td>SpeechAct</td>
<td>Action</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>Term</td>
<td>Term</td>
<td>Term</td>
<td>Term</td>
<td>Term</td>
<td>Term</td>
<td>Term</td>
</tr>
<tr>
<td>Concept</td>
<td>Concept</td>
<td>Concept</td>
<td>Concept</td>
<td>Concept</td>
<td>Concept</td>
<td>Concept</td>
<td>Concept</td>
</tr>
<tr>
<td>Ontology (trm)</td>
<td>Knowledge-Component</td>
<td>Knowledge-Component</td>
<td>Ontology (true)</td>
<td>Ontology (true)</td>
<td>Ontology (true)</td>
<td>Ontology (true)</td>
<td>Ontology (true)</td>
</tr>
<tr>
<td>Local (trm)</td>
<td>Environment</td>
<td>Environment-Zone</td>
<td>Environment</td>
<td>Environment</td>
<td>Environment</td>
<td>Environment</td>
<td>Environment</td>
</tr>
<tr>
<td>Analysis</td>
<td>Inhabit</td>
<td>Inhabit</td>
<td>Inhabit</td>
<td>Inhabit</td>
<td>Inhabit</td>
<td>Inhabit</td>
<td>Inhabit</td>
</tr>
<tr>
<td>Outlay</td>
<td>Resources</td>
<td>Resources</td>
<td>Resources</td>
<td>Resources</td>
<td>Resources</td>
<td>Resources</td>
<td>Resources</td>
</tr>
</tbody>
</table>

Note 2: There are several agent architectures missing from the table that were a part of the analysis (i.e. Prometheus and GoalNet) for space reasons.

Note 3: There are others that were not a part of the original analysis either for reasons of scale (e.g. PASSI [4], MaSE), or those within technologies only, e.g. Fischer et al [7] have extracted the agent meta-model behind the JACK agent platform and dubbed it JackMM (and AgentMM in an earlier paper [12]), and the meta-model from the JADE agent platform and dubbed it JadeMM.

Note 4: The DigitalFriend V1 tool [10] is an implementation of the ShadowBoard agent architecture [9].

Our motivation for collecting and comparing agent meta-models was as the primary input of agent concepts as entities into a Normalisation process, to arrive at a well-formed agent meta-model. So our initial interest in the comparison was analytic only. However, others are also currently endeavouring to define a large all-inclusion agent meta-model, with larger agendas than ours. For example Hahn et al [12] demonstrate the usefulness of the MDA (model driven architecture) approach to software development under the auspices of OMG [17]. MDA includes PIF (Platform Independent Model) and PSM (Platform Specific Model) models, such that an application specification at a PIF level, can then be transformed into the PSM of an existing technology – both PIFs and PSMs are meta-models. Hahn et al take an existing complete PIF from the SOA field called PIM4SOA, they present a meta-model representing the agent platform JACK, and then they demonstrate how one maps to the other. In a later paper Fischer et al [7] further present the meta-model from the JADE platform, and again demonstrate the transformation process from PIM4SOA to the PSMs for both JADE (JadeMM) and JACK (JackMM). They then discuss the use of a model-to-text transformation using the MOF-Script language that allows them to transform from PSM meta-models to an actual executable JACK project, and an executable JADE project.

From that demonstration Fischer et al then propose that a unified agent meta-model (dubbed PIM4Agents) is a worthy goal, should be obtainable, and could provide interoperability between many of the current disparate agent meta-models, methodologies and technology platforms. The concepts within Service Orient Computing (SOA) are well defined and less complex than AO concepts, so that attaining a PIM4Agents is likely to be a much harder task than it was to build the existing PIM4SOA, but a rewarding one.

4. Future Work

There are some hurdles to cross with meta-model comparisons and transformations as noted in Section 3, specifically when the concepts across models do not quite match up, however we are investigating the MDA process. We can see significant benefit in putting the ShaMAN meta-model into a MDA compatible PSM format in the near future, particularly if the AO community arrives at a well-represented PIM4Agents as discussed in Section 3 above. That would allow MDA application specifications to be automatically scripted into an executable project in our evolving platform (currently dubbed DigitalFriend V2), in the way that Fischer et al have already demonstrated with PIM4SOA, JADE and JACK.
References
