A MDA Approach for Agent-oriented Development using FAML

Carlos Eduardo Pantoja and Ricardo Choren
Military Institute of Engineering, Pça Gen Tibúrio 80, Rio de Janeiro, 22270-290, RJ, Brazil

Keywords: MDA, Agent-oriented Development, Multi-agent Systems.

Abstract: There are several multi-agent development modelling tools to aid system design and implementation. However, such tools are modelling language specific, which imposes the use of a given modelling technique. This paper presents a MDA approach for multi-agent system development based on the FAML meta-model. Thus the approach aims to support a wide range of modelling techniques.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

Software agents are cognitive and autonomous components, situated in an environment, which are capable of flexible and autonomous actions in this environment in order to achieve their projects goals (Wooldridge, 2000). The multi-agent systems (MAS) approach allows complex systems modelling (Bellifemine et al., 2007).

Currently, there are several agent-oriented modeling techniques like Tropos (Bresciani et al., 2003), Prometheus (Padgham and Winikoff, 2004), among others. Besides, there are some development platforms that generate code from a specific agent-oriented modeling techniques, such as PDT (Sun et al., 2010), which generates code from Prometheus to JACK and IDK (Gomez-Sanz et al., 2008), which generates from INGENIAS to JADE. To overcome the issue of specific code generators for specific modeling languages, it is important to employ meta-models. FAML (Beydoun et al., 2009) is a meta-model that unifies common concepts of existent agent-oriented modeling languages, thus being a potential candidate for standardization and engineering a MAS System using the Model-Driven Development techniques.

This paper proposes a MDA approach for agent-oriented development that provides a set of explicit transformation rules in Query, View, Transformation Language (QVT) to construct a Platform Specific Model (PSM) and generate code to the Jason agent-oriented programming language. This paper is structured as follows: section 2 the describes some base concepts; section 3 shows the proposed MDA approach; section 4 presents a simplified working example of the MDA approach. Section 5 describes some and section 6 concludes the paper.

2 MODELING MAS

This section describes the FAML meta-model, the Jason programming language and the concepts used to develop the transformations, mappings and templates for the proposed approach.

2.1 FAML

FAML is a meta-model that unifies different agent-oriented modelling languages inside the same software engineering domain for MAS development. This generic meta-model was validated to guarantee the concepts promoted by the extant agent-oriented methodologies (Beydoun et al., 2009).

The FAML is divided into two scopes: the design-time, where the central concept is the agent's system, and the runtime, where the central concept is the environment where the agent is situated. An agent is also a central concept in the meta-model, defining an internal and external scope. Thus, four levels can be raised: system; environment; agent, and; agent definition levels.

The internal scope of an agent comprises the agent and the agent-definition levels. The first level is responsible for the description of the agents' definitions and specifications. The second is responsible for the runtime agent-internal classes, describing the mental state of an agent, its plans actions, roles and communications.

The external scope of an agent comprises the

Copyright © 2012 SCITEPRESS (Science and Technology Publications, Lda.)

system and the environment-definition levels. The system level describes the system organizations and its relationships (roles and tasks). The environment level describes all agents situated in the environment, its resources and facets. Furthermore, the environment also maintains an event history and is generated by a related system.

2.2 QVT and MTT

QVT is an OMG approach to describe model-to-model transformations. It is used since queries can be applied in a source model. Moreover, the view is a description of the target model and the transformation is the part where the results of queries are projected into the view, creating the target model.

QVT can be used to create platform specific models (PSM) in the MDA viewpoint. Model To Text (MTT) is an OMG approach to transform models to text artefacts, e.g. code (OMG, 2008). This work uses the Acceleo (Obeo, 2012) tool that uses QVT and MTT. It was projected to allow source code generation from UML and MOF.

2.3 Jason

Jason is a programming language for MAS development. In Jason an agent can be programmed based on beliefs, goals, plans and actions. The agents have a belief base, which is changed in consequence of their perceptions about the environment. A plan has a trigger event and whenever a belief is changed, a sequence of actions can be started. In Jason, an agent also has committed goals that represent a pursued project goal (Bordini et al., 2007).

However, Jason does not support organizational programming. Moise+ (Hubner et al., 2002) is an organizational model that can be integrated to Jason to support the specification of an agent organization in three dimensions: structural specification, functional specification and deontic specification. Moreover, Jason does not provide environmental support. JaCaMo (Boissier et al., 2012) is used to enhance Jason with environmental concerns.

3 THE PROPOSED APPROACH

This section proposed the MDA approach for agent-oriented development using FAML and Jason (figure 1). The approach intends to be modelling language independent thus it begins its model transformation from FAML concepts. Any agent modelling language that adheres to FAML (Beydoun et al., 2009) can be used. The approach provides a set of transformations to map instances of FAML concepts to the target model based on Jason constructions, in order to generate MAS code.

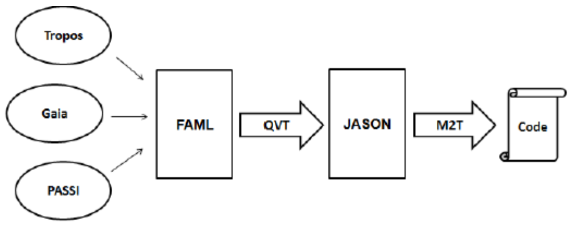


Figure 1: The proposed MDA approach.

To design the transformations, it was necessary to make some extensive modifications in FAML. For example, originally, in FAML, an agent can send several messages in a communication. Jason does not have the communication concept, so each communication will have only one message. Another modification is the unification of the Agent Goal and System Goal into a single Goal class, since in Jason (integrated with Moise+ and JaCaMo), an agent shares the same goals with the System. This unification decreases the number of mappings performed by the approach. The modified meta-model can be seen in figure 2.

The Jason model (figure 3) will be populated from the results of the QVT transformations from the FAML source meta-model (see figure 4). The main transformation called PimToPsm receives the FAML model and returns a Jason model completely mapped. The main mapping recovers all FAML model objects and calls FamlToGeaplam.

The initial mapping transforms a FamlMM object into a geaplamMM. It maps all the FAML systems objects to Jason systems objects calling the SystemToSystem mapping, which transforms roles, organizations, tasks and environments from FAML to roles, groups, missions and environments of Jason/Moise+/JaCaMo platform.

The RoleToRole transformation maps the agentgoal concept from FAML to Jason and calls itself recursively to determine the roles hierarchy. The SystemGoalToGoal transformation only maps the objects attributes. The OrganizationToGroup transformation determines the groups' hierarchy and recovers the roles already mapped previously. The TaskToMission transformation maps the agentgoal from FAML to Jason and maps the objects attributes from task to mission.

The EnvToEnv maps all agents from FAML to Jason and maps all the facets to percepts calling the FacetToPercept. The AgentToAgent transformation

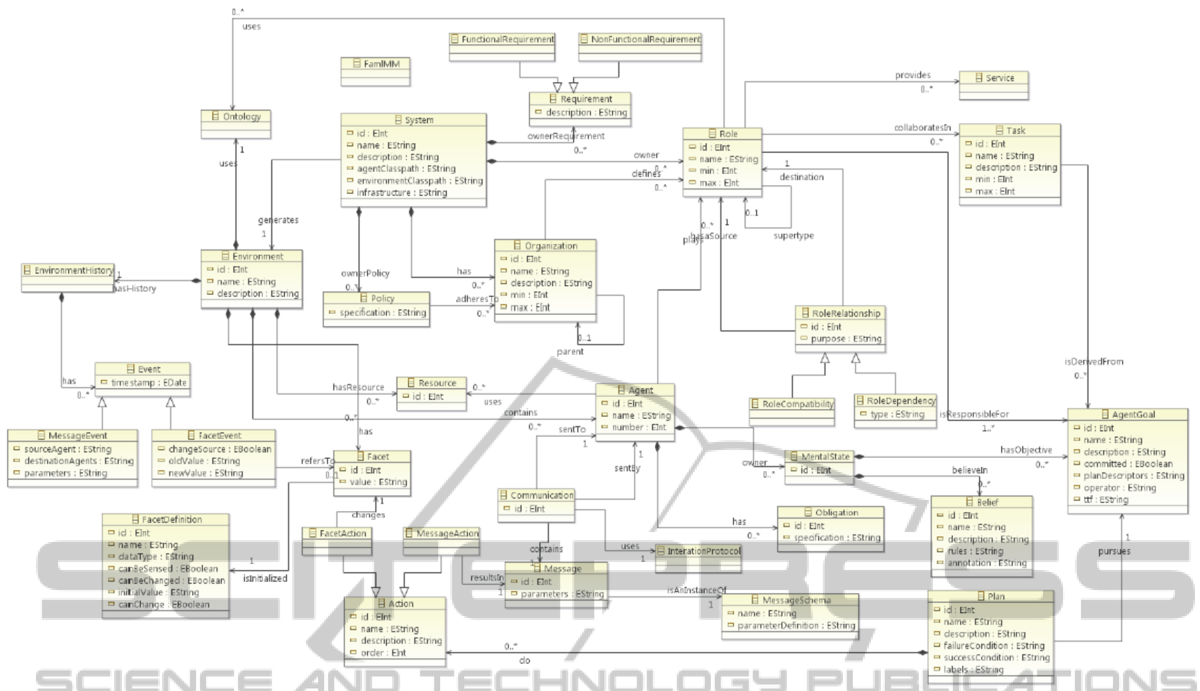


Figure 2: The modified FAML metamodel.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

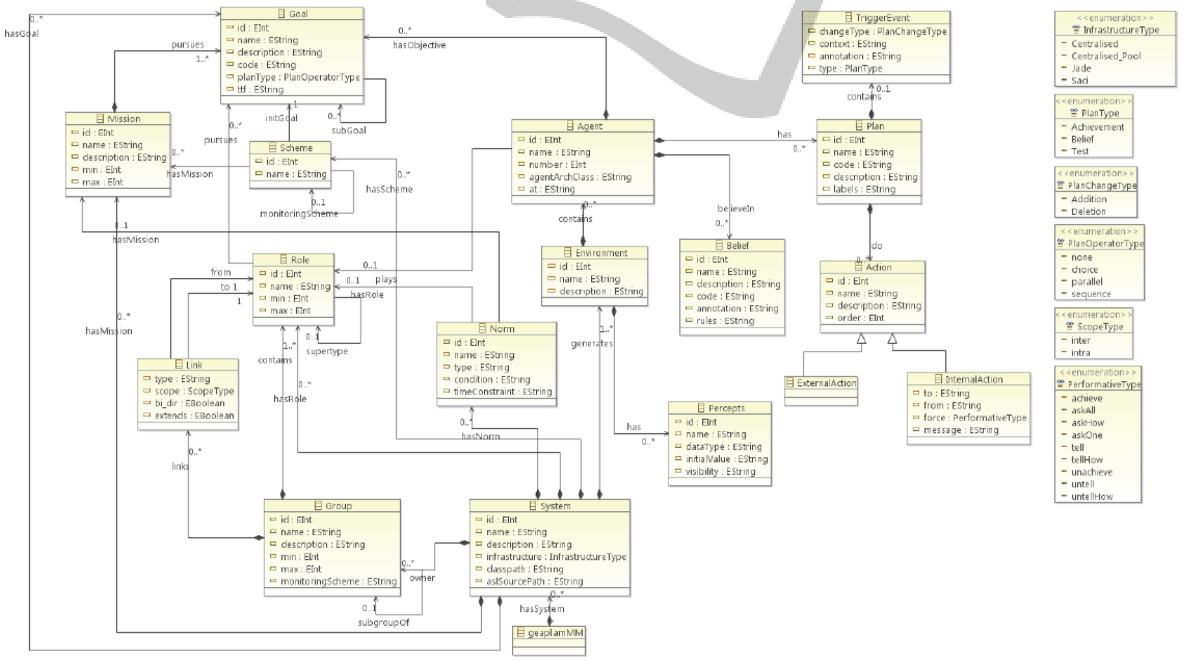
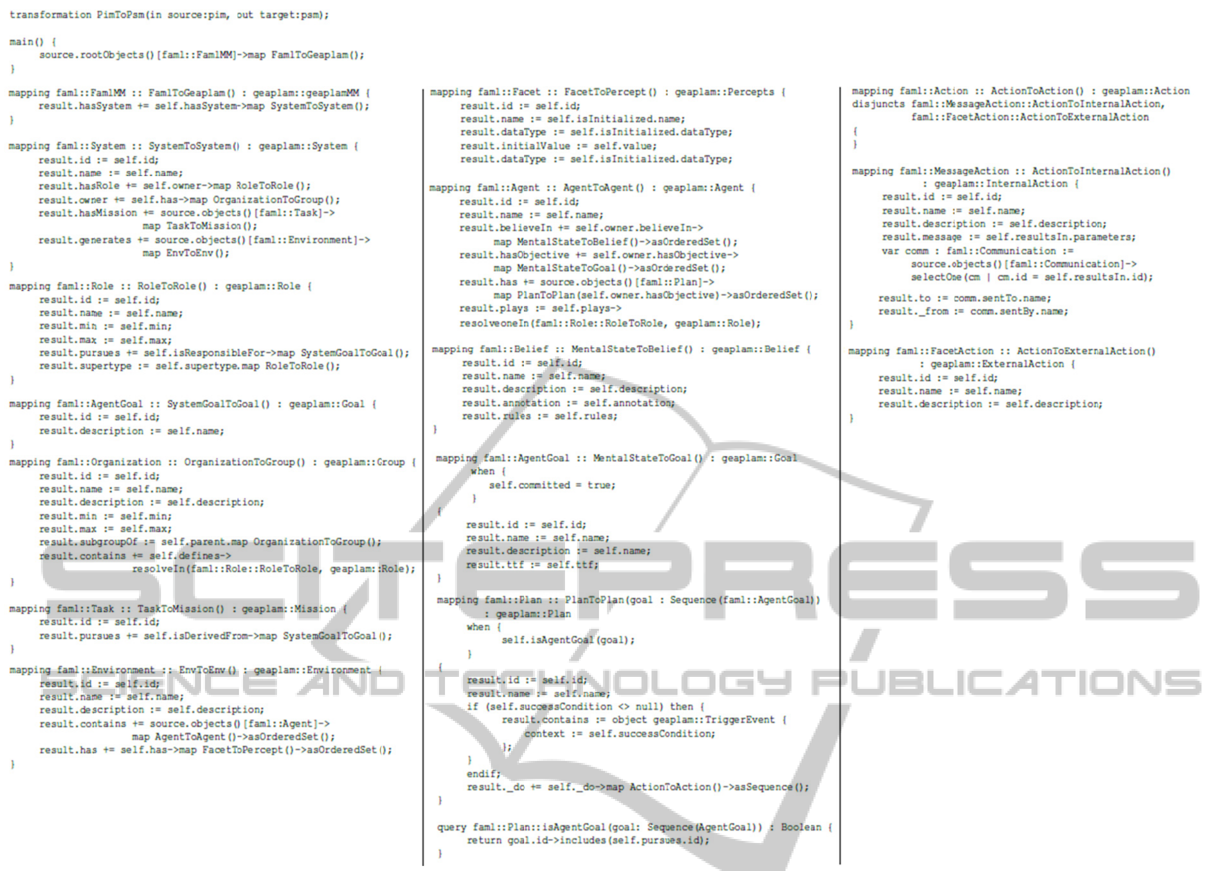


Figure 3: The target model based on Jason, Moise+ and JaCaMo.

maps a MentalState into Beliefs and Goals, in case of the attribute committed is true. The transformation also maps Plans and Roles that an agent can play. The Beliefs are mapped by the MentalStateToBelief transformation. The Goals of an agent is mapped with the MentalStateToGoal transformation, whenever the attribute committed is true.

The PlanToPlan transformation first checks the existence of an associated goal pursued by a Plan. The FAML plan is mapped into two structures: the Plan and the TriggerEvent. The transformation also maps the Actions of an agent Plan. The



```
transformation PimToPsm(in source:pim, out target:psm);

main() {
     source.rootObjects()[faml::FamlMM]->map FamlToGeaplam();
}

mapping faml::FamlMM :: FamlToGeaplam() : geaplam::geaplamMM {
     result.hasSystem += self.hasSystem->map SystemToSystem();
}

mapping faml::System :: SystemToSystem() : geaplam::System {
     result.id := self.id;
     result.name := self.name;
     result.hasRole += self.owner->map RoleToRole();
     result.owner += self.has->map OrganizationToGroup();
     result.hasMission += source.objects()[faml::Task]->
                      map TaskToMission();
     result.generates += source.objects()[faml::Environment]->
                      map EnvToEnv();
}

mapping faml::Role :: RoleToRole() : geaplam::Role {
     result.id := self.id;
     result.name := self.name;
     result.min := self.min;
     result.max := self.max;
     result.pursues += self.isResponsibleFor->map SystemGoalToGoal();
     result.supertype := self.supertype.map RoleToRole();
}

mapping faml::AgentGoal :: SystemGoalToGoal() : geaplam::Goal {
     result.id := self.id;
     result.description := self.name;
}

mapping faml::Organization :: OrganizationToGroup() : geaplam::Group {
     result.id := self.id;
     result.name := self.name;
     result.description := self.description;
     result.min := self.min;
     result.max := self.max;
     result.subgroupOf := self.parent.map OrganizationToGroup();
     result.contains += self.defines->
                 resolveIn(faml::Role::RoleToRole, geaplam::Role);
}

mapping faml::Task :: TaskToMission() : geaplam::Mission {
     result.id := self.id;
     result.pursues += self.isDerivedFrom->map SystemGoalToGoal();
}

mapping faml::Environment :: EnvToEnv() : geaplam::Environment {
     result.id := self.id;
     result.name := self.name;
     result.description := self.description;
     result.contains += source.objects()[faml::Agent]->
                 map AgentToAgent()->asOrderedSet();
     result.has += self.has->map FacetToPercept()->asOrderedSet();
}
```

```
mapping faml::Facet :: FacetToPercept() : geaplam::Percepts {
     result.id := self.id;
     result.name := self.isInitialized.name;
     result.dataType := self.isInitialized.dataType;
     result.initialValue := self.value;
     result.dataType := self.isInitialized.dataType;

mapping faml::Agent :: AgentToAgent() : geaplam::Agent {
     result.id := self.id;
     result.name := self.name;
     result.believeIn += self.owner.believeIn->
          map MentalStateToBelief()->asOrderedSet();
     result.hasObjective += self.owner.hasObjective->
          map MentalStateToGoal()->asOrderedSet();
     result.has += source.objects()[faml::Plan]->
          map PlanToPlan(self.owner.hasObjective)->asOrderedSet();
     result.plays := self.plays->
     resolveoneIn(faml::Role::RoleToRole, geaplam::Role);

mapping faml::Belief :: MentalStateToBelief() : geaplam::Belief {
     result.id := self.id;
     result.name := self.name;
     result.description := self.description;
     result.annotation := self.annotation;
     result.rules := self.rules;
}

 mapping faml::AgentGoal :: MentalStateToGoal() : geaplam::Goal
      when {
         self.committed = true;
      }
 {
      result.id := self.id;
      result.name := self.name;
      result.description := self.name;
      result.ttf := self.ttf;
 }

 mapping faml::Plan :: PlanToPlan(goal : Sequence(faml::AgentGoal))
        : geaplam::Plan
      when {
           self.isAgentGoal(goal);
      }
 {
      result.id := self.id;
      result.name := self.name;
      if (self.successCondition <> null) then {
           result.contains := object geaplam::TriggerEvent {
               context := self.successCondition;
           };
      }
      endif;
      result._do += self._do->map ActionToAction()->asSequence();
 }

 query faml::Plan::isAgentGoal(goal: Sequence(AgentGoal)) : Boolean {
      return goal.id->includes(self.pursues.id);
 }
```

```
mapping faml::Action :: ActionToAction() : geaplam::Action
disjuncts faml::MessageAction::ActionToInternalAction,
          faml::FacetAction::ActionToExternalAction
{
}

mapping faml::MessageAction :: ActionToInternalAction()
           : geaplam::InternalAction {
      result.id := self.id;
      result.name := self.name;
      result.description := self.description;
      result.message := self.resultsIn.parameters;
      var comm : faml::Communication :=
           source.objects()[faml::Communication]->
           selectOne(cm | cm.id = self.resultsIn.id);

     result.to := comm.sentTo.name;
     result._from := comm.sentBy.name;
}

mapping faml::FacetAction :: ActionToExternalAction()
           : geaplam::ExternalAction {
     result.id := self.id;
     result.name := self.name;
     result.description := self.description;
}
```

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Figure 4: The proposed transformation set.

ActionToAction makes the proper action transformation to FacetAction or MessageAction. If it is a Message Action, the ActionToInternalAction transformation maps an Internal Action from the FAML Communication and Message. Otherwise, the ActionToExternalAction transformation maps the objects attribute into an External Action.

It is important to mention that the FAML Ontology, EnvironmentHistory, Event, MessageEvent, FacetEvent, InterectionProtocol, MessageSchema, Resource and Obligation concepts were not used in any transformation.

4 A SIMPLE EXAMPLE

In this section, an example of a MAS application, modelled in Prometheus (Padgham andWinikoff, 2004), called Domestic Robot (Bordini et al., 2007). The domestic robot system (figure 5) contains three agents: the robot, the owner and the supermarket. The agent robot is responsible for the goal of delivering beer to its owner. When the robot receives a message from the agent owner, it goes to the fridge, takes a bottle of beer and brings it back to the owner. The Robot still manages the beer stock and some consumption rules. The agent owner has only the goal of drink beer. The agent supermarket delivers more beer in response to the agent robot in case of low stock.

The QVT transformations were implemented using the M2M framework for Eclipse platform. All transformations and mappings were modularized to increase the maintainability and possible future improvements in the related meta-models.

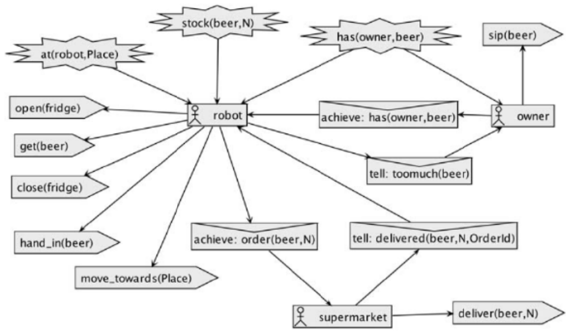
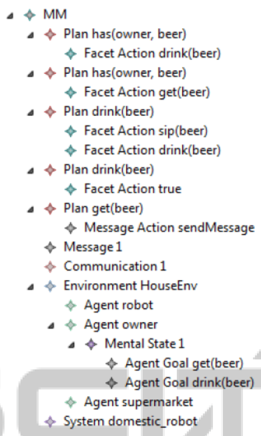


Figure 5: The Domestic Robot System model.

The first step of the approach is to instantiate the FAML meta-model based on the Domestic Robot System. It can be done importing a XML file or creating it using the Ecore tree viewer. The FAML instantiated objects can be seen in figure 6.



```
MM
  Plan has(owner, beer)
    Facet Action drink(beer)
  Plan has(owner, beer)
    Facet Action get(beer)
  Plan drink(beer)
    Facet Action sip(beer)
    Facet Action drink(beer)
  Plan drink(beer)
    Facet Action true
  Plan get(beer)
    Message Action sendMessage
  Message 1
  Communication 1
  Environment HouseEnv
    Agent robot
    Agent owner
      Mental State 1
        Agent Goal get(beer)
        Agent Goal drink(beer)
    Agent supermarket
  System domestic_robot
```

Figure 6: The FAML instance of the System.

The second step of the approach is to execute the transformation, populating an instance of the PSM meta-model from the source model, the FAML model, and generate the code. It can be done in two manners: directly from PIM to code or transforming from PIM to PSM and afterwards, generate the code. The advantage of the second approach is that it allows the modeller to adjust the instance of the PSM meta-model before generate the code. The domestic robot.mas2j file was generated as follows:

```
MAS domestic_robot {
    infrastructure: Centralised
    environment: HouseEnv
    agents:
        owner;
        supermarket agentArchClass SupermarketArch;
        robot;}
```

The agent object is composed of plans, beliefs and goals. In the agent owner example, it only has plans and a goal. The model to text then generates all agents' files. The owner.asl file was generated as follows:

```
get(beer).

+get(beer): true <-
    .send(robot, achieve, has(owner, beer)).

+drink(beer): has(owner, beer) <-
    sip(beer);
    !drink(beer).

+drink(beer): not has(owner, beer) <-
    true.

+has(owner, beer): true <-
    !drink(beer).

-has(owner, beer): true <-
    !get(beer).
```

The object environment is composed by agents and percepts. The model to text then generates the environment classes. The HouseEnvModel.java was generated as follows:

```
import jason.environment.grid.GridWorldModel;

public class HouseEnvModel extends GridWorldModel {

    protected HouseEnvModel(int arg0, int arg1, int arg2) {
           super(arg0, arg1, arg2);
         // TO DO
    }
}

import jason.asSyntax.*;
import jason.environment.*;

public class HouseEnv extends Environment {

    @SuppressWarnings("unused")
    private HouseEnvModel model;

    @Override
    public void init(String args) {
         model = new HouseEnvModel(0, 0, 0);
         updatePercepts();
    }

    void updatePercepts() {
         //TO DO
    }

    public boolean executeAction(String ag, Structure action) {
         //TO DO
         return false;
    }

}
```

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

5 RELATED WORK

There are some works to generate agent code from agent-oriented models, e.g. (Sun et al, 2010) (Gomez-Sanz et al., 2008) (Cossentino, 2005). However, those are initiatives for specific modelling languages. The INGENIAS development kit is a tool to generate JADE (Bellifemine et al., 2007) code from INGENIAS models. The PDT tool generates JACK (Winikoff, 2005) code from Prometheus models. The PASSI toolkit proposes a stepwise refinement from requirements to code, compiling PASSI diagrams and generating JACK code.

Such tools do not use an agent-oriented meta-model: the transformations are done directly from the model element to a code piece. Thus, such tools are not prepared to support other modelling languages. The MDA approach presented in this paper relies on the FAML meta-model that is capable of representing meta-models of most existing MAS methodologies, including INGENIAS, Prometheus and PASSI (Beydoun et al. 2009). In this sense, the approach allows for the designer to choose the modelling language to use in the MAS development (regarding it is FAML-compliant).

They also do not use a platform meta-model.

Particularly, the PASSI toolkit does not even use a MDA approach: code generation is responsibility of an AgentFactory Tool. Thus they are not devised to evolve to support other agent-oriented platforms. The approach uses Jason meta-model since it is provides a BDI approach that can handle organization and environment aspects (through Moise+ and JaCaMo extensions). JADE is a Java-based platform that is simple and do not use BDI concepts. JACK is a BDI-enhanced JADE but it does not support other agent-oriented concepts.

6 CONCLUSIONS

This paper presented a MDA approach for MAS development, using a generic agent meta-model for agent-oriented modelling languages for Jason's code generation. The approach realizes a set of transformations in QVT language to populate the program language model. Afterwards, texts artefacts are generated from templates using the model to text language.

The paper also presented an example of the Domestic Robot System modelled in Prometheus. The FAML meta-model is instantiated and transformed to the Jason Model. The set of transformations were implemented using the Model To Model do EMF. The code was generated using the Model to Text language and was implemented with Acceleo.

The use of MDD techniques provides a standardization of the development, increasing the maintainability and re-usability of the models and facilitating the developers to transform the modelled project into code. For future works, a modelling environment with graphical tools for instantiate the FAML meta-model will be developed using GMF. The FAML can be used to generate code for others agent-oriented program languages like JACK, JADE and JADEX.

REFERENCES

Bellifemine, F., Caire, G., and Greenwood, D. (2007). Developing multi-agent systems with JADE. *Wiley series in agent technology*.

Beydoun, G., Low, G., Henderson-Sellers, B., Mouratidis, H., Gomez-Sanz, J. J., Pavon, J., and Gonzalez-Perez, C. (2009). FAML: a generic metamodel for MAS development. *IEEE Trans. Softw. Eng.*, 35(6):841–863.

Boissier, O., Bordini, R. H., Hubner, J. F., Ricci, A., and Santi, A. (2012). JaCaMo project. http://jacamo.sourceforge.net/.

Bordini, R. H., Hubner, J. F., and Wooldridge, W. (2007). Programming Multi-Agent Systems in AgentSpeak using Jason. *Jonh Wiley and Sons*, London.

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., and Perini, A. (2003). Tropos: An Agent-Oriented Software Development Methodology.

Cossentino, M. (2005). From Requirements to Code with the PASSI Methodology. In Sellers, H. B. and Giorgini, P., editors, Agent-Oriented Methodologies, *volume 3690 of LNCS*, pages 79–106. Idea Group Pub.

Gomez-Sanz, J. J., Fuentes, R., Pavón, J., and Garcia-Magarino, I. (2008). Ingenias development kit: a visual multi-agent system development environment. *In Proceedings of the 7th AAMAS*, p. 1675–1676.

Hubner, J. F., Sichman, J. S. a., and Boissier, O. (2002). A model for the structural, functional, and deontic specification of organizations in multiagent systems. *In Proceedings of the 16th SBIA*, p. 118–128.

Obeo (2012). Acceleo: MDA generator - home. http://www.acceleo.org/.

OMG (2008). MOF model to text transformation language, v 1.0.

Padgham, L. and Winikoff, M. (2004). Developing Intelligent Agent Systems: A Practical Guide. *John Wiley*.

Sun, H., Thangarajah, J., and Padgham, L. (2010). Eclipse-based prometheus design tool. *In Proceedings of the 9th AAMAS*, p. 1769–1770.

Winikoff, M. (2005). Jack intelligent agents: An industrial strength platform. In Bordini, R., Dastani, M., Dix, J., Fallah Seghrouchni, A., and Weiss, G., editors, *Multi-Agent Programming, volume 15 of Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 175–193. Springer US.

Wooldridge, M. (2000). Reasoning about rational agents. Intelligent robotics and autonomous agents. *MIT Press*.