Model Driven Engineering for Designing Adaptive Multi-Agent Systems (ESAW 2007)

Sylvain Rougemaille, Frédéric Migeon, Christine Maurel, Marie-Pierre Gleizes

Model Driven Engineering for Designing Adaptive Multi-Agents Systems

Sylvain Rougemaille, Frédéric Migeon, Christine Maurel, and Marie-Pierre Gleizes

IRIT – Paul Sabatier University – 118, Route de Narbonne
31062 Toulouse, Cedex 9, France
Tel.: +33 561 558456
{rougemaille,migeon,maurel,gleizes}@irit.fr

Abstract.
A challenge for our days is to provide new efficient CASE (Computer Aided Software Engineering) tools enabling MAS designers towards Model Driven Engineering (MDE) approaches. The goal of MDE is to improve the de-velopment process and the quality of the software produced. Our work focuses on two different aspects of MAS. The functional one, which is application de-pendent and close to the decision process of agents, and the operational one related to elementary capabilities of agents. For each point of view, we have de-fined specific meta-models. Our goal in this paper is to provide a mapping from the functional meta-model to the operational that constitutes a specific platform model. As we are interested in adaptive systems, we have to deal with adaptation both at the agent and the system level. We address this problem by respectively using the JavAct flexible architecture and the Adaptive MAS principles.

1 Introduction

A challenge for our days is to provide new efficient CASE (Computer Aided Software Engineering) tools enabling MAS designers towards Model Driven Engineering (MDE) approaches. The goal of MDE is to improve the development process and the quality of the software produced. Basically, Model Driven Architecture (MDA) [1] proposes to automatically generate the PSM (Platform Specific Model) by merging early specifications expressed as the PIM (Platform Independent Model) and some intermediary PM (Platform Model) by using several automated transformations.

The work presented in this paper consists of the enhancement of the existing ADELFE methodology [2], based on the AMAS (Adaptive Multi-Agent Systems) theory and dedicated to the design of self-organising systems. The aim is to enrich ADELFE with a development phase providing tools based on MDE. In particular, we want to focus on two aspects of a MAS: the functional aspect which is application dependent and close to the decision process of agents, and the operational related to elementary skills of agents. For each point of view, we have defined specific meta-models, we support mapping from the functional meta-model to the operational one avoiding the merging phase of a classical MDA approach.

Furthermore, systems we are interested in must be adaptive and we take into account the adaptation at both agent and system levels. We have developed the JavAct

agent-based middleware [3] whose agents are designed to be adaptive by the means of a component-based flexible architecture and which is a particularly well suited target for implementing an AMAS whose main principle is system self-adaptation. However, JavAct is used as an implementation platform for our work and is not in the scope of this paper. We simply use its adaptation capabilities thanks to its architecture and promote it at the model level.

In this paper, we advocate that the design of an agent must be realized by considering two different levels: an operational one and a functional one (Section 2). Once the proposed approach is positioned in relation to existing works (Section 3), different points of view of the meta-models, at the two levels previously determined, are described (Section 4). Then, the mapping enabling the transformation from a functional meta-model to an operational one is defined (Section 5). The paper ends with a discussion and a presentation of the perspectives of the proposed approach (Section 6).

2 Operational and Functional Adaptation

In living systems, adaptation is linked to the species' learning and evolution capabilities according to the Darwin's theory [4]. Thus, adaptation is a process enabling changing systems' structure and\or behaviour in order for the species to survive. In artificial systems, adaptation can occur as an off-line manner, for example when a programmer stops the code execution, changes a line of code and launches it again. But, the more complex and interesting concept of adaptation which is treated in this paper, is self-adaptation in which the artificial system performs some internal changes, during the execution to enable the system adaptation. The reason why an artificial system has to self-adapt is explained in [5]. According to the DARPA definition, Robertson and al. said: "self-adaptive software evaluates its own behaviour and changes behaviour when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible".

The systems that we are focusing on can be an agent or a MAS. So, to self-adapt an agent or a MAS has to change; i.e. to modify its behaviour or a part of its structure. For example, an artificial ant can learn that when it is in front of a wall, it has to turn (behaviour) or if it possesses the adequate skills, it can self-add one leg (adding a part). We can assume that, in general, the adaptation process starts by an interaction between the system and its environment. As Maturana and Varela have shown, there exists a coupling between the system and the environment [6], there exists a very close relation between a system and the environment in which it is executed. This relation is based on the system perceptions of its environment and the actions it makes to deal with. Once deployed, the system has to react to its surrounding environment stimuli, one way to react is, in particular, to self-adapt.

The designer must lead an analysis at the system (or global level), and at the agent (or local level) in order to design a self-adaptive MAS. Most of the works on multi-agent design agree that an agent follows the life cycle composed of three main steps: perceive, decide and act. In each step, the agent design must take into account two levels: an operational one and a functional one.

Considering the perceive-decide-act lifecycle, we understand those levels as follow. The operational level is made up of all the tools or means which enable the perception

of the environment and the performance of actions such as moving, sending messages. The functional level is related to all the means which enable the choice of an action to perform such as “if there is an obstacle in front of me I have to turn left or right”, i.e. the decision process. Those levels are detailed in the next sections.

2.1 Operational Point of View

The operational part of an application gathers everything which is independent of the application layer. Therefore, operational adaptation can be seen as updating the system to its execution environment, without altering its behaviour, so that it is more efficient or simply compliant with it. For example, an agent entering an unsecured zone could have its message sending protocol become encrypted, independently of its functional behaviour.

For this purpose, we developed the JavAct middleware [3] where agents benefit from a meta-level architecture. While the base-level contains the functional part of an agent, the meta-level gives flexibility to this agent by means of interconnected micro-components; each of them implements an operational skill of the agent (mobility, message sending, environment perception, dynamic creation etc.). Consequently, operational adaptation is obtained by changing a micro-component dynamically. Of course, operational adaptation can be triggered by functional concerns and is defined by the programmer at design time, i.e. concerns and conditions from the base level imply meta-level modifications. For example, the encrypted communication could be a small part of a complex and functional adaptation of an agent which perceives its environment as unsecured and decides that encryption is the best way to adapt.

2.2 Functional Point of View

The functional behaviour can be viewed as the result produced by a system in an applicative context. It is what the system can realize to achieve the requirements of an application and usually is domain dependent. MAS adaptation can be due to change either in individual behaviour of agents or in the collective behaviour of the system. In this paper, we focus on functional adaptation at the system level.

At the agent level, the function is related to the result provided by the agent action and self-adaptation consists in changing the decision process leading to another action, or learning a new action to be done. The adaptation of its behaviour can be realized by endowing the agent with a learning process. With this ability, the agent can act differently and can adapt its behaviour to subsequent events in its environment by using the knowledge learnt from making previous decisions. The research community on agent learning has produced a lot of work such as case-based reasoning, reinforcement learning, neural networks, etc. which are usually used in the MAS community.

System adaptation consists in changing the behaviour of the collective in response to new or modified environments. In the AMAS (Adaptive Multi-Agent Systems) approach [7], we have worked to highlight a generic behaviour which enables the system to self-adapt. The AMAS theory provides a guide to design self-organising systems. It is based on the observation that modifying interactions between the agents of the system modifies also the global function and makes the system self-adapt to changes in its environment. The local criterion used by agents to decide which kind of

action they perform, is the cooperative attitude. An agent behaviour is led by the two following attitudes: a repairing one and an anticipative one. If no change appears in its environment, an agent performs its nominal behaviour and if changes occur it analyses the situation and chooses the most cooperative action related to its environment in which other agents evolve.

According to the AMAS theory, every agent pursues an individual objective and interacts with agents it knows by respecting cooperative techniques which lead to the avoidance of Non Cooperative Situations (NCS) like conflict or concurrence. Faced with a NCS, a cooperative agent acts to come back to a cooperative state and permanently adapts itself to unpredicted situations while learning from others. The AMAS theory is based on how an agent can avoid failures and this approach is an exceptions handling mechanism at the agent level.

2.3 Adaptation Levels

As we have seen in this section, we have identified different kinds of adaptation but also different levels of concerns. The following table sums up the idea:

	Functional Adaptation	**Operational Adaptation**
Agent	Classical learning approaches	JavAct
System	AMAS approach	

Operational adaptation in JavAct mainly concerns the agent itself whereas the AMAS approach fits functional adaptation of the system. However, it is easy to obtain operational adaptation of the system by coordinating operational adaptation of agents as well as functional adaptation of agents (such as learning for example), which can be obtained from AMAS concerns. The operational mechanisms involved in the execution of an AMAS are at least, constituted by the set of the mechanisms used by each type of agent. The complexity of the operational adaptation mechanism of the whole system would depend on the heterogeneity of the agents of which it is composed. Each agent is responsible of its own operational adaptation, thus the system operational adaptation only depends on its agents. Furthermore, the consistency should be guaranteed by cooperation rules that insure for example the understanding of messages.

So, our aim is the providing of means to design self-adaptive MAS which enable us to take into account in one tool the operational adaptation at the agent level and functional adaptation at the system level. We have developed the ADELFE method [8], to design AMAS but this method consists only of the three first steps of the design life cycle: Requirements, Analysis and Design Workflows. Our aim is to add the development phase to ADELFE by taking into account the two previous levels of adaptation. Because the objective is to reduce the design duration and the complexity of the task for designers, they should only focus on the system functional adaptation (NCS definition) and the agent definition while the operational adaptation should be automatically handled by the JavAct middleware thanks to model transformations. Mapping AMAS operational concerns to JavAct specific architecture (see section 5) is the purpose of the presented work and is the first step to reach that goal.

3 Related Work

Currently, most of the existing agent-based methodologies [9], [10] have fully taken into account the first phases of a software development life cycle: requirements, analysis and design phases. The phases such as implementation, test, deployment and maintenance are more or less treated in the following well-known methods ADELFE [8], INGENIAS [11], PASSI [12], and TROPOS [13]. But an effort has been made on these phases, notably in proposing new tools to facilitate code generation. In this section, the main works using models transformation in order to design MAS are reviewed and the main tools coming from MDE are analysed.

3.1 MAS Related Work

The first result to improve the agent-based software development is about meta-models definition in MAS methods and the second is about the use of model transformations coming from the MDA or MDE community in order to generate automatically the code for a given platform.

3.1.1 MAS Meta-models

Since 2003, initiated by FIPA (Foundation for Intelligent Physical Agents) working groups, several meta-models have been defined. The difficulty was to find a unique and agreed meta-model, so several meta-models have been defined.

AALAADIN [14] is a meta-model based on agent, group and role concepts. It enables principally the description of organisation; agents belong to groups in which they handle roles. Agent are intentionally not detailed, thus, developers are free to choose the one that better fit their requirements. A concrete adoption of AALAADIN is used in the MadKit platform [14] in the requirements and design phases.

FAML [15] meta-model is a meta-model built to take into account every kind of existing MAS. It is expressed in two layers: design-time and runtime, concerning both the agent and the system point of views. At design level, the expressed concepts are role, task, agent, plan, action, ontology and environment access points. At runtime, environment, events, system access points, plans, action, message, desires, beliefs and intention. At each level, the relationships between concepts are explained.

GAIA [16] meta-model highlights the notions of: role (which is refined in responsibility, activity, permission concepts), agent, communication (with protocols), organisation (structure and rules) and environment (resource).

INGENIAS [11] meta-model integrates different results on multi-agent and agent works. By consequence, it considers a MAS from five complementary viewpoints:

- Organization (workflow, group, agents, roles, resources, and applications),
- Agent (tasks, goals, mental states and roles),
- Tasks/goals which describes their decomposition and the consequences of their execution in terms of: mental entity, interaction, resource, application,
- Interactions (agent, goal, role, task, specification),
- And environment (agent, application, resource).

PASSI meta-model [12] is composed of three domains: the solution, the agency and the problem domains. The problem domain deals with the user's problem in terms of scenarios, requirements, ontology and resources. In the agency domain, the main concepts are agent, role, task and communication with AIP message. The implementation domain describes the structure of the code solution in the chosen FIPA-compliant implementation platforms.

TROPOS [13] is a method organized in five phases: early requirements, late requirements, architectural design, detailed design and implementation. The main concepts enabling expression of intentional and social concepts are: actor, goal, plan, resource and the relationships between them such as actor dependency goal decomposition, plan decomposition, means-end and contribution relationships.

Some attempts have been made such as the gathering of ADELFE, GAIA and PASSI meta-models [17] but the meta-model obtained was seen as being to complex so that the authors didn't pursue this work. It seems better to define different meta-models in relation with the type of MAS to be designed. However, these works on meta-models have given us a better understanding of the concepts used in the MAS community and lead us to the use of MDA or MDE tools.

3.1.2 MAS and MDA/ MDE

Few works on MAS engineering have integrated tools coming from MDE, and the most advanced are: INGENIAS[1], MetaDIMA [18] and TROPOS [19].

MetaDIMA helps the designer to implement MAS on the DIMA platform using MetaGen which is a MDE tool dedicated to the definition of meta-models and models. DIMA is a development and implementation platform developed in Java where agents are seen as a set of dedicated modules (perception, communication etc.). MetaDIMA provides a set of meta-models and a set of knowledge-based systems on top of DIMA to ease the design of MAS by providing languages more specific than Java code.

In TROPOS (see 0), all the phases use different models which are described by meta-models; it also uses UML notation and automatic transformations. For example, it translates plan decomposition into a UML 2.0 activity diagram by using a transformation language based on the following three concepts: pattern definition, transformation rules and tracking relationships.

INGENIAS proposes to transform the MAS expressed in the INGENIAS meta-model in code dedicated to a given platform using the modelling language of INGENIAS and the implementation model of the platform. Its main originality consists in providing evolutionary tools. Because tools used for transforming specification in code are based on meta-models, if the meta-model specifications evolve the tools can also evolve.

Our work pursues the same objective as the works described previously although it addresses the adaptation issue from both system and agent points of view. In fact, we aim at taking it into account and providing design and generation tools to implement such adaptive systems. For this purpose, we propose to generate an adapted execution platform for AMAS, using MDE tools and principles as well as the flexibility of the JavAct middleware.

[1] http://grasia.fdi.ucm.es/ingenias/

3.2 MDE Tools

This section presents a brief overview of tools we have already used or foresee using in the model driven scope.

3.2.1 Model Editing Tools

The eclipse IDE provides the *EMF* (Eclipse Modeling Framework) [20] which provides a meta-modelling language called *Ecore* (allowing description of meta-models) and features to edit, handle and modify models. On top of this plug-in, we use *Topcased* [21] as an *Ecore* editor and a graphical editor generator, i.e. we define editors for our different modelling languages thanks to its generative capabilities. With the same purpose, we have compared *Topcased* to *GMF* (Graphical Modeling Framework) which possesses more or less the same functionalities while adopting a different approach for graphical editor description where each aspect of the editor is described by a model.

3.2.2 Transformation Tools

We plan to use model to model transformations to implement our mapping (see section 5.) and model to text transformations to generate JavAct code. Those transformations have to be supported by tools and languages. We mainly focus on *ATL* (Atlas Transformation Language) [22] and *Kermeta* [23] which both provide tools based on *EMF* and are implemented as Eclipse plug-in. *ATL* is a hybrid language providing declarative features while *Kermeta* is defined as a meta-programming language close to OO programming languages. However, we plan to use their specificities respectively to implement transformations and to equip our modelling language with execution capabilities (for simulation purpose).

4 Meta-models

The main idea of our work can be summed up as follow. On the one hand we have the AMAS theory which intrinsically implies to deal with self-adaptation and on the other one, we have the JavAct middleware whose agents are designed to be flexible and adaptive (on the operational point of view). Our purpose is to bridge the gap between them as automatically as possible. To achieve this task we assume to tackle this problem at the highest abstraction level, i.e. at meta-model level, and use model transformations to build that bridge.

To describe both JavAct agent model and AMAS in a model driven approach, we have developed two dedicated modelling languages (DSML[2]) which are themselves described by meta-models. Those meta-models can be seen as representations of the main concepts and relationships we have identified for each of these particular domains (AMAS and JavAct micro-architecture). Our idea is to map automatically agents from an AMAS model to an adapted JavAct micro-Architecture model; this could be done using model transformation languages as presented in the previous section. Thus, it is necessary to describe as precisely as possible what are the key concepts of the two domains. The following sections give a brief overview of those meta-models.

[2] Domain Specific Modeling Language.

4.1 AMAS Meta-model

The AMAS meta-model was elaborated from the ADELFE meta-model [17] enriched by three distinct logical points of view to describe an AMAS. Each of them represents a specific part of the AMAS theory on which we want to put emphasis:

- *System point of view*: it is devoted to the description of the system and its surroundings in terms of *Entities* which populate it (perceptible objects of the "world").
- *Agent point of view*: this part of the meta-model represents agent internal characteristics.
- *Cooperation point of view*: it represents taxonomy of *Non-Cooperative Situations* an AMAS agent is likely to encounter.

In the context of this paper, we focus on the agent point of view.

4.1.1 AMAS Agent Point of View

An AMAS agent is made up of various modules, parts, managing a sector of its activities and life-cycle. Typically, the AMAS agent life-cycle is defined according to the three phases: *perception*, *decision* and *action*. From these phases and the needs they imply in terms of environmental interactions, knowledge representation, non-cooperative situations avoidance, etc. we determine the following meta-model concepts:

- Environmental interactions are represented by *Perception* and *Action* on the *Entities* as well as the means to carry them out (*Actuator*, *Sensor*). *CommunicationAction* consists of direct interaction with other agents by the means of messages (*Message*) whose protocol is defined in the *System* point of view (*CommunicationProtocol*).
- *DecisionModule* gathers the *Aptitudes* and the *CooperationRules* enabling an agent to determine the next actions to lead. This decision is taken according to agent knowledge and aptitudes as well as its cooperation rules which propose actions to overcome possibly detected *NonCooperativeSituations (NCS)*. Without NCS detection, an agent carries out its local and nominal function determined by its aptitudes.
- *Knowledge* represents what an agent possesses. It has self-*Representations* and *Representations* of the medium surrounding it (Environment *Entities,* agents of the system). It also possesses *Skills* and its *Characteristics* possibly perceptible by other agents of the system (*isPerceptible*).

4.2 Micro-architecture Description Language (µADL) Meta-Model

µADL is a micro-architecture description language based on the previously expounded principle of JavAct micro-architecture [6] (cf. section 2). It focuses on operational mechanisms definition of JavAct agents in terms of micro-components. A new micro-component assembly is called a micro-architecture and it constitutes a new agent "style" which can be used in a JavAct application. Thus, µADL provides the concepts of *MuArchitecture*, *MuComponent* and of *Interface* (see Figure 1). A *MuArchitecture*

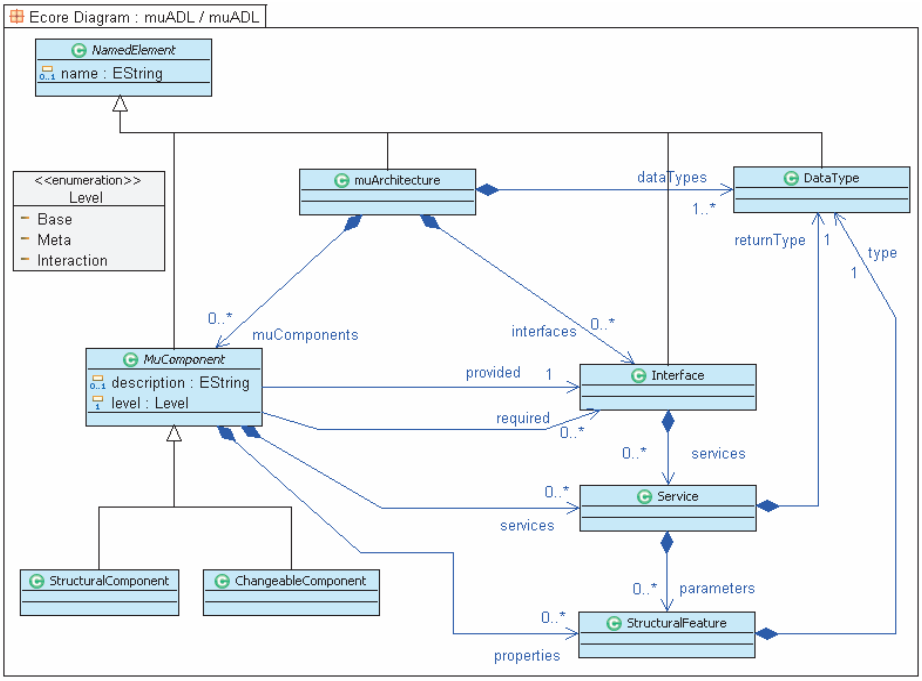

Fig. 1. Micro-Architecture meta-model

is a composition of *MuComponents* connected to each other by the fact that they provide or require *Interfaces*. We have defined two different *MuComponents*:

- *StructuralComponents*: corresponding to the micro-components that are not intended to be redefined or to be substituted. According to the policy defined in JavAct, they represent core concepts of the architecture that are used to achieve reflection and delegation mechanisms.
- *ChangeableComponents*: conversely, they correspond to the micro-components which could be replaced, modified, extended and so on.

Each JavAct micro-component is defined by the following fields:

- *name*: its name,
- *description*: a short informal description of its operating properties,
- *level*: the level in which it offers its services (*base, meta, interaction*),
- *properties*: data that could be necessary to achieve a particular service,
- *services:* the *Services* it implements.
- *provided*, *required*: the *Services* it requires and it offers to other *MuComponents* through *Interfaces*.

StructuralFeatures and *DataType* are both meta-classes that enable the definitions of *MuComponent* properties which can be assimilated to typed attibutes in the Object world.

JavAct micro-architectures are forced by the fact that they always have to delegate services provided by the base level *µComponents* to the agent functional code. It implies that the micro-architectures contain at least one *Controller* and one *meta-access* component. This kind of constraint is verified with dedicated OCL rules.

5 The Mapping Process (Mapping AMAS Agents to the JavAct Platform)

To instantiate a particular JavAct agent model for each agent within the AMAS model, we have to map concepts from the AMAS meta-model to µADL ones. At this time, we simply focus on environmental interactions of AMAS agent (i.e. Action, Perception, etc.). Furthermore, we present an "informal" mapping that could be implemented with model transformation languages.

5.1 Meta-models Mapping

The mapping takes place between the AMAS and the µADL meta-models, so we have to specify which concepts have to be mapped to each others. Thus, the aim of this mapping is to operationally adapt JavAct agent so that they become compatible with the AMAS theory, that is to say: bring them to a higher degree of environmental interaction and cooperation. This can be done by expressing intrinsic characteristics of AMAS agents in terms of *MuComponents*. In other words, we focus on the operational aspects of AMAS agents and we try to describe them as a micro-architecture. To express more conveniently this mapping, we provide an overview table which represents µADL concepts for each AMAS meta-class (see Table 1).

Each *muComponent* has to provide at least one *Interface* containing at least one *Service* of those implemented by the *muComponent*. This is a generic mapping rule: we consider that *Service* providing is necessarily done through those *muComponent* related interfaces.

Most of the AMAS meta-classes map to *MuComponents*, although there are exceptions to this rule. Those exceptions are related to particular implementation choices we made. To illustrate those choices, consider the *Knowledge* meta-class and subclasses. Knowledge is a particularly important part of a cooperative agent because of its implication in the decision process; thus, we decide to reify this concept in term of a *MuComponent*. This *MuComponent* is related to two more specific *Skill* and *Characteristic MuComponents*, which are designed to embody respectively all skills and characteristics of cooperative agents as vectors.

Another important point of interest is the *MuComponent* level, which specifies the scope of a *MuComponent*, i.e. whether it can be accessed by the functional code (*base* level) or not (*meta* level). The *Interaction* level represents *MuComponents* dedicated to agent external interactions. For example, the *Action perform()* service is used to define reaction to *NCS*, which is the purpose of AMAS agent functional code; thus we define *Action* as a *base* level *MuComponent*.

As we define a meta-level mapping between AMAS and µADL meta-models, the next section presents what should be the result of a transformation at model level. This is done using a simple and well-known AMAS example and focusing on environmental interactions.

Table 1. Mapping AMAS meta-classes to µADL concepts

AMAS		µADL					
		MuComponent					
Meta-class name		Name	Level	Services	Provided	Required	Properties
Action	□ Maps to □	Actions	base	performAction()	performAction()	perform()	-
							actionList
Communication Action		Communication	base	send() receive()	send() receive()	-	protocol
Message							messageType
Perception		Perception	meta	perceive()	perceive()	perceive()	-
Actuator		Actuator	interaction	Enabled actions	Enabled actions	-	-
Sensor		Sensor	interaction	Enabled perceptions	Enabled perceptions	-	-
Knowledge		Knowledge	base	update() getRepresentation()	update() getRepresentation()	update() getRepresentation()	
Representation							repList (element)
Skill		Skills	meta	getAction() update() getRepresentation()	getAction() update() getRepresentation()		
							skillList (element)
Characteristic		Characteristics	meta	update() getRepresentation()	update() getRepresentation()		
							characteristicList (element)
Decision Module		Decision	meta	decide()	decide()	getRepresentation() getAction()	
Agent		LifeCycle	base	run()	run()	perceive() decide() perform()	

5.2 Model Mapping: Ants Example

This example comes from the ANTS project [24], whose purpose was to define software ant-robots based on ethological observations concerning the foraging process of an anthill. The aim was not to simulate the real process but use the available information

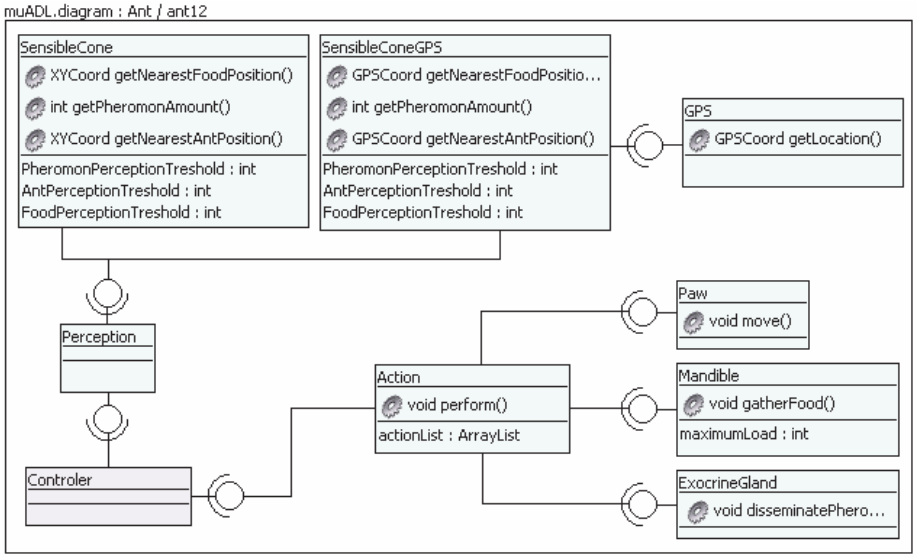

Fig. 2. Part of an ant agent µADL graphical model (informal mapping result)

coming from the biologists to implement robots which have to collect distributed resources in an unknown environment.

In this part, we focus on the environmental interactions described with the AMAS meta-model and what should be the result of the mapping in the µADL meta-model. In other words, the µADL model specifies the appropriate operational mechanisms of a JavAct cooperative ant-agent.

From an AMAS point of view, an ant agent possesses a *SensibleCone* which allows it to perceive food, pheromones and other ants of the colony. This sensor is qualified by properties limiting its scope (threshold area values for food, pheromones and ants). Ants are also able to explore their environment (thanks to *move action*), to gather food when they find some and if the load is not too heavy (*gatherFood action*) and to disseminate pheromones on their way back to the nest (*disseminatePheromon action*). All those actions are carried out by actuators. By focusing on this part of the AMAS ant colony model and applying our mapping we obtain the following µADL model (see Figure 2), in which all actuators have been mapped to micro-components (*Paw*, *Mandible* and *ExocrineGland*) as sensor has been (*SensibleCone*).

JavAct micro-architecture enables dynamic operational adaptation by switching two micro-components provided that they implement the same interface. Figure 2 presents a static vision of that capacity (µADL model). At runtime it is possible for an ant JavAct agent to choose the more appropriate way to "sense" its environment. For instance, a robot ant evolving on a real tangible terrain should use the *SensibleConeGPS* micro-component in order to locate itself and its surroundings. But if the GPS device is damaged JavAct enables the ant agent to switch the useless component with the *SensibleCone* micro-component which uses a simulated environment.

6 Conclusion and Perspectives

In this paper, we present our work whose aim is to add the development phase to ADELFE by considering two adaptation levels: a functional one for the system, and an operational one for JavAct agents. As the goal is to reduce the duration and the complexity of the design of AMAS, we investigated a MDE approach; this is based on model transformations in order to facilitate code production for a given platform.

As we aim to provide JavAct agent version which be automatically fitted to the Adaptive Multi-Agent System (AMAS), we have developed two dedicated modelling languages (DSML) with meta-models which describe respectively JavAct agent and AMAS.

As we have seen, adaptation is the central point of our methodology and tools. The designer will be able to describe functional adaptation in models which conform to the AMAS meta-model. The mapping described in the previous section will then result in a model of a specific JavAct agent architecture. This model can be considered as a Platform description Model (PM) adapted to the nature of the application. Therefore, the gain is two-fold. On the one hand, we combine functional adaptation of AMAS and operational adaptation of JavAct agents. On the other one, we simplify the complexity of design: details of implementation are hidden during early conception phases. In continuation of this work, we are studying the different stages for implementing a concrete prototype of the CASE tool. As was mentioned in section 3.2, we have already used Topcased or GMF tools to generate automatically graphical editors for the DSML defined. We hope now to be able to automate the production of these editors from meta-models such as the AMAS one. Of course, this cannot be done from the AMAS meta-model only. Some additional information must be collected in order to automate to the generation of the editors.

Finally we have also some experiences in the production of JavAct code using an ad-hoc generator we developed. This tool, called Agent$^{\varphi}$, allows generation of JavAct code from an assembly description. However, work has still to be done for generalizing its use and for integrating it in a MDE tool.

Acknowledgments

We would like to thank Carole Bernon, Thierry Millan and Pierre Glize for discussions about the meta-models.

References

[1] OMG, MDA Guide, Object Management Group, Inc., Final Adopted Specification (2003)
[2] Bernon, C., Gleizes, M.-P., Peyruqueou, S., Picard, G.: ADELFE, a Methodology for Adaptive Multi-Agent Systems Engineering. In: Petta, P., Tolksdorf, R., Zambonelli, F. (eds.) ESAW 2002. LNCS (LNAI), vol. 2577. Springer, Heidelberg (2003)
[3] Leriche, S., Arcangeli, J.P.: Adaptive Autonomous Agent Models for Open Distributed Systems. In: International Multi-Conference on Computing in the Global Information Technology (ICCGI 2007), March 2007, pp. 19–24. IEEE Computer Society, Los Alamitos (2007)

[4] Darwin, C.: On the Origin of Species by Means of Natural Selection. John Murray, London (1859)
[5] Robertson, P., Laddaga, R., Shrobe, H.: Introduction: the First International Workshop on Self-Adaptive Software. In: Robertson, P., Shrobe, H.E., Laddaga, R. (eds.) IWSAS 2000. LNCS, vol. 1936, pp. 1–10. Springer, Heidelberg (2001)
[6] Varela, F., Maturana, H.: The Tree of Knowledge: The Biological Roots of Human Understanding. Shambhala Press, Boston (1998)
[7] Capera, D., Georgé, J.-P., Gleizes, M.-P., Glize, P.: The AMAS Theory for Complex Problem Solving Based on Self-organizing Cooperative Agents. In: Proc. 12th IEEE International Workshops on Enabling Technologies, Infrastructure for Collaborative Enterprises, Linz, Austria, June 9-11, pp. 383–388. IEEE Computer Society, Los Alamitos (2003)
[8] Bernon, C., Gleizes, M.-P., Picard, G.: Enhancing Self-Organising Emergent Systems Design with Simulation. In: International Workshop on Engineering Societies in the Agents World (ESAW 2006), Dublin (September 2006)
[9] Bergenti, F., Gleizes, M.-P., Zambonelli, F. (eds.): Methodologies and Software Engineering for Agent Systems. Kluwer, Dordrecht (2004)
[10] Henderson-Sellers, B., Giorgini, P. (eds.): – Agent-Oriented Methodologies. Idea Group Pub. (June 2005)
[11] Gomez Sanz, J., Fuentes, R.: Agent Oriented System Engineering with INGENIAS. In: Fourth Iberoamerican Workshop on Multi-Agent Systems, Iberagents 2002 (2002)
[12] Cossentino, M.: From Requirements to Code with the PASSI Methodology. In: Henderson-Sellers, B., Giorgini, P. (eds.) Agent-Oriented Methodologies, June 2005, pp. 79–106. Idea Group Pub. (2005)
[13] Giorgini, P., Kolp, M., Mylopoulos, J., Castro, J.: Tropos: A Requirements-Driven Methodology for Agent-Oriented Software. In: Henderson-Sellers, B., Giorgini, P. (eds.) Agent Oriented Methodologies, pp. 20–45. Idea Group (2005)
[14] Gutknecht, O., Michel, F., Ferber, J.: The MadKit Agent Platform Architecture, Research Report, LIRMM (April 2000)
[15] Beydoun, G., Gonzalez-Perez, C., Henderson-Sellers, B., Low, G.: Developing and Evaluating a Generic Metamodel for MAS Work Products. In: Garcia, A., Choren, R., Lucena, C., Giorgini, P., Holvoet, T., Romanovsky, A. (eds.) SELMAS 2005. LNCS, vol. 3914, pp. 126–142. Springer, Heidelberg (2006)
[16] Cernuzzi, L., Juan, T., Sterling, L., Zambonelli, F.: The Gaia Methodology: Basic Concepts and Extensions. In: Bergenti, F., Gleizes, M.-P., Zambonelli, F. (eds.) Methodologies and Software Engineering for Agent Systems. Kluwer Academic Publishers, Dordrecht (2004)
[17] Bernon, C., Cossentino, M., Gleizes., M.-P., Turci, P., Zambonelli, F.: A study of some Multi-Agent Meta-Models. In: Giorgini, P., Mueller, J.P., Odell, J. (eds.) The Fifth International Workshop on Agent-Oriented Software Engineering (AOSE 2004), New York, USA, July 19 (2004)
[18] Guessoum., Z., Jarraya, T.: Meta-Models & Model-Driven Architectures, Contribution to the AOSE TFG AgentLink3 meeting, Ljubljana (2005)
[19] Perini, A., Susi, A.: Automating Model Transformations in Agent-Oriented Modelling. In: Proceedings of 6th International Workshop AOSE 2005, Utrecht, NL, July 25-26 (2005)
[20] Budinsky, F., Steinberg, D., Ellersick, R.: Eclipse Modeling Framework: A Developer's Guide. Addison-Wesley Professional, Reading (2003)

[21] Farail, P., Gaufillet, P., Canals, A., Camus, C.L., Sciamma, D., Michel, P., Crégut, X., Pantel, M.: TOPCASED project: a Toolkit in OPen source for Critical Aeronautic SystEms Design. In: Embedded Real Time Software (ERTS) (2006)
[22] Jouault, F., Kurtev, I.: Transforming Models with ATL. In: Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005, Montego Bay, Jamaic (2005)
[23] Muller, P., Fleurey, F., Jézéquel, J.: Weaving Executability into Object-Oriented Meta-Languages. LNCS, Montego Bay, Jamaica. Springer, Heidelberg (2005)
[24] Topin, X., Fourcassié, V., Gleizes, M.-P., Théraulaz, G., Régis, C., Glize, P.: Theories and experiments on emergent behaviour: From natural to artificial systems and back. In: Proceedings on European Conference on Cognitive Science, Siena (1999)