Towards Agent-Oriented Model Driven Engineering

Iván García-Magariño[1], Sylvain Rougemaille[2], Rubén Fuentes Fernández[1], Frédéric Migeon[2], Juan Pavón Mestras[1], and Marie-Pierre Gleizes[2]

[1] D. Software Engineering and Artificial intelligence
Facultad de Informatica
Univesidad Complutense Madrid, Spain
{ivan_gmg, juan.pavon, ruben}@fdi.ucm.es
[2] Institut de Recherche en Informatique de Toulouse
SMAC Team
Université de Toulouse, France
{rougemai,migeon,Marie-Pierre.Gleizes}@irit.fr

Abstract. This work gathers the experience of two different research groups that developed the agent-oriented methodologies *INGENIAS* and *ADELFE*. The particular features of the agent paradigm makes that the application of MDE approaches to it has to consider certain issues that are not common in mainstream Software Engineering. In particular, MAS methodologies usually consider several abstraction levels and perspectives, producing a richer modelling than for instance object-oriented approaches. Besides, the agent research is still object of important evolution with changing and growing conceptual frameworks. The view of these MAS research groups is promoting model transformations to bridge these gaps between perspectives in modelling, and to support the change in the modelling languages and their corresponding tools. This focus on transformations should also allow taking advantage of the semantic richness of MAS concepts in code generation, producing software that is closer to the final product since the meaning of the concept and its related behaviour are more precise.

1 Introduction

Developing *Multi-Agent Systems* (MAS) in the scope of modern information systems is a complex activity. Different proposals have been made to alleviate the designer's work, mainly through methodologies and their support tools. Among them, we are particularly involved in the methodologies *INGENIAS* [23] and *ADELFE* [3]. Both of them share a software engineering approach based on Model Driven Engineering (MDE). We have gathered our experience in the definition of each methodology and the development of their tools in order to highlight the benefits and challenges of this approach for MAS development. Of course, synthesis presented here gains also from related works which are quickly surveyed before conclusion. However, our true background, and precise technical use of MDE, is limited to the scope of *INGENIAS* and *ADELFE*, which therefore constitute the focus of the paper.

The paper analyzes the particular requirements of Agent Oriented Model Driven Engineering (AOMDE), term that we propose to express the application of MDE to the development of MAS. These requirements mainly rise from the need of dealing with rich, heterogeneous, and changing conceptual frameworks, which makes necessary to consider the evolution of the modelling languages and their related tools, the links between different perspectives, and also the possibility of carrying out sophisticated processes over the models that take advantage of that conceptual richness.

The organization of the paper is as follows. Section 2 describes the works that have been used as the basis for this paper. It begins describing the principles of MDE and then introduces the *INGENIAS* and *ADELFE* agent-oriented methodologies. Section 3 discusses the application of MDE in the scope of our methodologies by focusing on meta-modelling and model transformations. This application has shown benefits of the approach but also limitations that Section 4 discusses. Section 5 broadens the discussion about agent-oriented methodologies and MDE with some related work, the methodologies Tropos [5] and Prometheus [21], that also adopts a model-driven approach. Finally, Section 6 draws some conclusions about the concept of AOMDE and the future work that agent researchers must consider in its application.

2 Background

2.1 Model Driven Approach

Model Driven Engineering (MDE) [17] is a software development approach that considers models as first class citizens. Model transformations are the means to automate the life-cycle from early design tasks to the implementation. Likewise, maintenance of systems can benefit from keeping a correlation between models and code.

Specifically, the OMG *Model Driven Architecture* (MDA) [17] is one of the most broadly promoted model-driven approaches. Its principles involve the UML standard [18] and a set of automatic processing languages to produce code. Starting with a requirements model called the *Computation Independent Model* (CIM), an abstract *Platform Independent Model* (PIM) is created. The idea is to produce automatically a *Platform Specific Model* (PSM) thanks to model transformations. A mapping between the PIM and the PSM is created according to the *Platform Model* (PM) and implemented as a transformation. The specific model is then used to generate the application code.

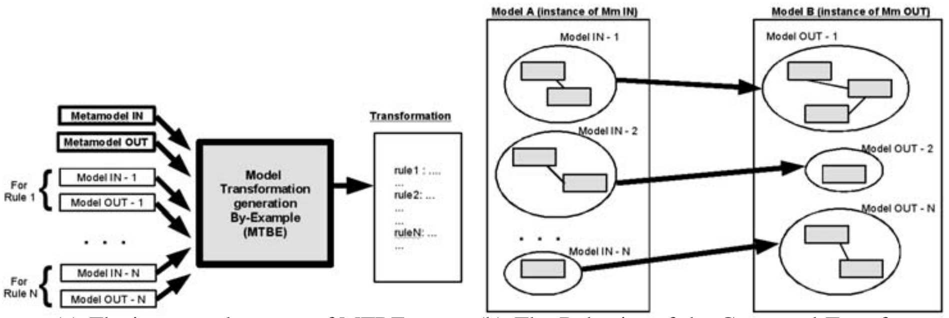
The main advantage of the MDA approach is that whenever the platform supporting the application changes, the abstracts models do not have to be modified. The transformations between PIM and PSM are the only part that have to be specified again.

As a recent initiative, Model transformation By-Example (MTBE) [32] is defined as the automatic generation of transformations from source and target model pairs. The common steps of MTBE are the following:

1. *Manual set-up of prototype mapping models*. The transformation designer assembles an initial set of interrelated source and target model pairs.
2. *Automated derivation of rules*. Based upon the available prototype mapping models, the transformation framework should synthesize (see Figure 1(a)) the set of model transformation rules. These rules must correctly transform (see Figure 1(b)) at least the prototypical source models into their target equivalents.
3. *Manual refinement of rules*. The transformation designer can refine the rules manually at any time. However, MTBE recommends these modifications to be included

in the pairs of models, so the alterations are not overwritten the next time the transformation is generated.

4. *Automated execution of transformation rules*. The transformation designer validates the correctness of the synthesized rules by executing them on additional source-target model pairs as test cases.



(a) The inputs and outputs of MTBE (b) The Behavior of the Generated Transformation

Fig. 1: Description of the Model Transformation By-Example (MTBE)

The MTBE approach overcomes the hard-coding of transformations, which frequently hinders the principles of MDE. MTBE follows MDE principles because its main products are models and transformations. In addition, transformation designers in MTBE do not need to learn a new model transformation language; instead they only use the concepts of the source and target modelling languages.

2.2 INGENIAS and the IDK

The *INGENIAS* methodology for the development of MAS was founded on the use of meta-modelling technique [23]. It covers the whole development cycle, from analysis to implementation, and provides tool support with the *INGENIAS Development Kit (IDK)*. The user defines with the IDK the specification of the MAS, which is the basis of its development. Figure 2 shows some relevant elements of the *INGENIAS* notation. To create this specification, the user has the *specification editor* of the IDK. This editor works as a host for plugins that format documentation, verify specifications and generate code. The key plugin for code generation is the *INGENIAS* Agent Framework (IAF) that produces code for the JADE platform [2] according to an architecture designed in INGENIAS that supports, among several features, deliberative and reactive agents, and a simplified management of protocols. This code generator has been introduced previously in the literature in [10, 9]. Therefore the IDK provides a way to develop MAS following the principles of *Model-Driven Development* [25]. The user defines the specification with the IDK Editor. This specification represents the model, which is the basis for developing the MAS.

This MDE approach provides to the *INGENIAS* methodology a robust and quick technique for the development of MAS. As a matter of fact, *INGENIAS* and the IDK have been applied successfully in several areas; for instance, in surveillance [24], knowledge management system [30] and social simulation [22].

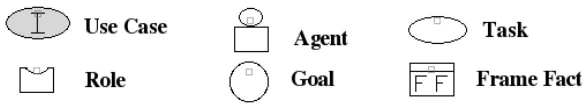


Fig. 2: The Most Relevant *INGENIAS* Notation.

2.3 ADELFE

ADELFE [3] defines an agent-oriented methodology for designing Adaptive Multi-Agent Systems (AMAS) [6]. *ADELFE* is mainly involved in the specification of systems intended to deal with complex environment dynamics and self-adaptation. The *ADELFE* methodology [3] aims at guiding AMAS designers through a development process based on the standard Unified Process (UP) [11] for object-oriented methodologies.

ADELFE has been recently improved with the use of UML 2.0 for the specification of interactions [1]. It also integrates a new model-driven implementation phase based on specific modelling languages and model transformations [28]. Therefore, the last *ADELFE* version covers the whole software development process, from early requirements to the implementation.

AMAS-ML (AMAS Modelling Language) is used in several steps of the *ADELFE* design phase, from the detailed agent structure design to the definition of its cooperative behaviour. The resulting model is used as an input for the implementation phase. By means of several model transformations, this phase allows generating the code of both the agents behaviour and the specific API supporting them [29]. Therefore, *ADELFE* provides a *MDE* process : from an abstract model defined with *AMAS-ML* it generates the application code.

3 Model-Driven from experience

3.1 Meta-modelling

MDE is based on models. Models have to be specified conforming to one meta-model. This meta-model could be considered as the abstract syntax of a modelling language. Both *INGENIAS* and *ADELFE* defines their own meta-models and each one are used as a basis for Domain Specific Modelling Languages (DSML).

[3] ADELFE ("Atelier de Développement de Logiciels à Fonctionnalité Emergente") is toolkit for designing softwares with emergent functionalities. It was a French RNTL-funded project (2000-2003)

ADELFE proposes two meta-models corresponding to specific modelling languages : *AMAS-ML* and *μADL*. *AMAS-ML* focuses on general AMAS specification and *μADL* in a specific architecture.

AMAS-ML formalizes concepts of the AMAS theory. It provides modelling elements to support the design of cooperative agents. A cooperative agent is specified as made up of various modules. Each module manage an aspect of its activities and life-cycle. Typically, the AMAS agent life-cycle is defined according to three main phases: perception, decision and action. The *AMAS-ML* meta-model considers the needs in these stages in terms of environmental interactions, knowledge, representation, non-cooperative situations avoidance, etc. For example, to carry out its perception and action phases, an AMAS agent requires environmental interactions. They are represented by perception and action on entities (elements of the environment) as well as the means to carry them out (actuator and sensor). These concepts appear in specific modules of the agent and are mandatory elements of *AMAS-ML* models. Thus, it ensures that designed agents are aware of their execution environment.

One important point for AMAS agents is their cooperative behaviour. In *AMAS-ML*, the decision module is a container for rules that describe the agent behaviour. Each rule is triggered by a particular state of the agent and implies several actions. The states are defined as logical conditions expressed over the agent knowledge. With *AMAS-ML* designer, the user can define the behaviour of cooperative agents and consequently, the emergent behaviour of the system.

μADL defines a flexible agent architectural style [15]. An architectural style [16] of a software unit is a set of rules and constraints that specifies its composition and evolution features. Here, software units are agents. These agents conform to an architectural style derived from the architecture of the agents in JavAct, which is a Java middleware for programming adaptive mobile agents[4] developed by the SMAC research team. Its main characteristic is the separation of concerns between the functional level related with the agents' behaviour and the operating level related with the internal mechanisms necessary for the execution of the functional code. This separation of concerns is provided at the operating level *via* fine-grained software components called micro-components. Those micro-components are connected to a mediator in a star-like topology according to the design pattern Mediator [8]. *μADL* is a language for micro-architecture modelling based on the architectural concepts previously explained. It guaranties several structural properties at design time, like consistency of service invocation between components or respect of the topology. It provides concepts such as micro-architecture, micro-component and interfaces for the connection between the micro-components and the *Mediator*. Thus, a micro-architecture consists in a micro-components assembly defined by means of the interfaces they provide to the *Mediator*. Any *μADL* model is an abstraction of a micro-architecture, that is, a type of agent, and is the source of a code generation process.

The *INGENIAS* meta-model was originally defined with the *Graph Object Property Relationship Role* (GOPRR) [13] meta-modelling language, which was supported by the MetaEdit+ tool. However, GOPRR did not have a standard notation and its support has decreased in comparison with other meta-modelling alternatives. For this reason,

[4] `http://www.irit.fr/PERSONNEL/SMAC/arcangeli/JavAct.html`

the *INGENIAS* meta-model was migrated to ECore. Since the *INGENIAS* meta-model is large, it contains 88 objects, 89 relationships and 98 roles, the Grasia group defined an automatic translation *GOPRR-to-ECore* for the meta-modelling support in the IDK. The remaining drawback, is that existing models that are conform to the GOPRR version of the *INGENIAS* meta-model, have to be also translated. An experimental implementation of the IDK 2.7 can serialize with instances of both GOPRR and ECore meta-models enabling model transformations according to the meta-model translation.

The use of meta-models in these methodologies facilitates the creation and the maintenance of design tools; either by being a preliminary tool architecture definition (in the scope of *INGENIAS*) or by being the basis for editors generation (in the scope of *ADELFE*). This point is specially significant in MAS domain. As a matter of fact, MAS concepts are still evolving, since no consensus has been yet reached on what should be called an agent or not. Meta-models constitute a flexible means to integrate new concepts in the design without changing the whole tool.

Meta-modelling helps to define precisely some concepts that are parts of modelling languages. Moreover, we have defined our meta-models using the wide spread Eclipse meta-modelling language ECore. It constitutes a "de facto standard" with many model driven tools compliant with it (ATL, GMF, GEF, EMF, etc.).

Although meta-models facilitate the evolution of design tools, maintaining models conformance whenever the meta-models change is still a challenging problem. In our experience, this constitutes a problem that is not well solved in MDE. Both *INGENIAS* and *ADELFE* meta-models are starting points for tool definition or generation. However, some parts have to be manually developed because of the advanced functionality they imply.

3.2 Model to Model transformations

Model-to-model transformations can be useful for different tasks in the Agent Oriented Software Engineering. Some of these usages currently done in *ADELFE* are:

- To integrate various modelling languages in the same frame. In particular, *ADELFE* uses the UML sequence diagram for the design of interaction protocols. The messages exchanged in these protocols are automatically integrated to the AMAS-ML model within the cooperative agent specification by means of a transformation.
- To separate different concerns. The *ADELFE* implementation uses a specific language, *μADL* (micro-Architecture Description Language) [29] to express operating mechanisms of agents (similar to non-functional concerns). The behaviour of cooperative agents, defined in the the *AMAS-ML* model, is separated of the operating concern. This concern consists in all the concepts that are involved in the creation and maintenance of the agent knowledge (perceptions, actions over environmental entities, and so on).

Model-to-model transformations are useful to gather good practises or designer experience. They allow to automate processes that have to be done by domain experts, as the translation of UML sequence diagrams into *AMAS-ML* communication actions. Designers who are not familiar with both domains can benefit from the knowledge embodied within this transformation. Moreover, these transformations guaranty that tasks are done in the way they have to be. For instance, *ADELFE* proposes to transform the *AMAS-ML* model into a specific *µADL* model. This transformation is the result of our experience on programming cooperative agents. We propose to separate the behavioural concerns (all that is related to the decision process) from the other part of the cooperative agents [29]. On one hand, it provides an abstraction level to the AMAS developer who can focus on the cooperative behaviour of the agent without getting involved, for instance, in the communication mechanisms. On the other hand, it allows assigning the development of fundamental services, such as communications, to developers who are not necessary AMAS experts. Therefore, this separation task is enforced within a model-to-model transformation that ensure the respect of our proposition. In *INGE-*

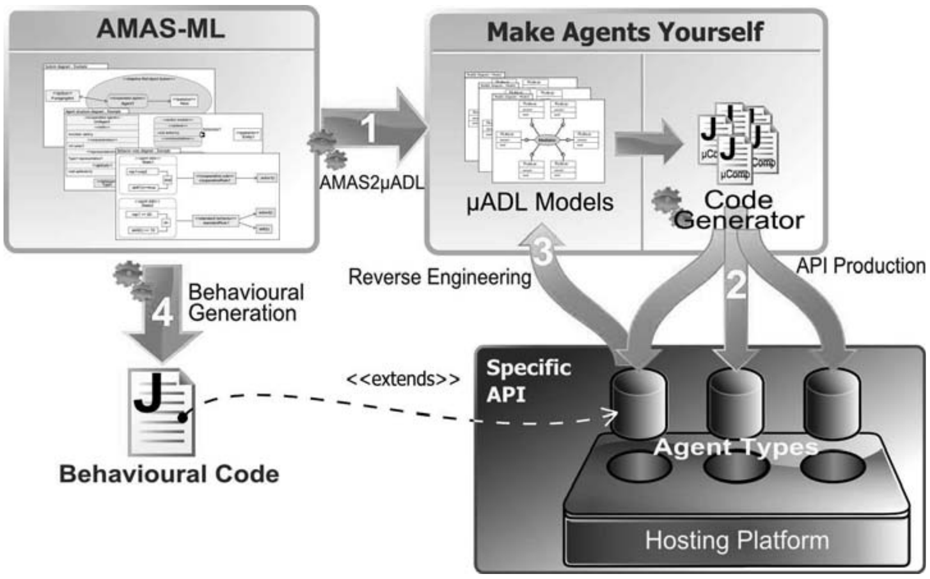


Fig. 3: General overview of *ADELFE* model driven implementation phase. Each arrow represents either a transformation or a generation step.

NIAS, model-to-model transformations are still not used. Currently, the code generation process uses templates. However, model-to-model transformations could be used as intermediate steps in the *INGENIAS* code generation. The Grasia research group is working on the application of model transformations for the improvement of meta-models, model transformations are planned to be created with an MTBE generator tool, whose prototype is available for practitioners (in Grasia web [5]). The application of this MTBE tool for *INGENIAS* is described later in this paper.

In the context of MAS, transforming agent concepts is more efficient because of the semantic of agent, MAS concepts raise the level of abstraction over implementation concepts or even Object Oriented concepts. The abstraction gap between Agent

[5] `http://grasia.fdi.ucm.es`

Oriented Modelling and Object Oriented Modelling makes difficult to map these two technical spaces. In our experience, model-to-model transformations are used to narrow this gap in several steps. Moreover, the automation of this task increase the productivity because model transformations are defined once and used several times.

3.3 Model to Text transformations

Model-to-text transformations make it possible to generate the programming code of systems from models. In the experience of both agent-oriented methodologies *INGENIAS* and *ADELFE*, these transformations have saved time and effort in academic and industrial projects. However, these transformations also have some drawbacks, such as maintaining the consistency between models and code.

There is a part of code that is very similar from one MAS to another and can constitute an architecture. For instance, the code related to triggering agents or the communication among them is very similar and frequently part of a platform. Although the extension of this code, varies from one platform to another, the automatization of this task has saved time and effort in all the agent-oriented technologies (JADE, JavaLeap, web services, etc. for *INGENIAS*; Java classes for *ADELFE*).

In *INGENIAS*, code templates are linked to the key concepts of *INGENIAS* meta-model. The information of models instantiates code-templates for the code generation. For some years, IDK generates code for several platforms: JADE, JavaLeap, Prolog, Servlets Soar. However, the improvements in both *INGENIAS* meta-model and the code generator, made it expensive to maintain the support for all the platforms. For this reason, at the current version of *INGENIAS*, only the JADE platform is supported.

In *ADELFE*, there are two different code generation aspects. The first one is providing a specific platform for the execution of the application code. This platform is based on flexible agent principles [14]. The generation of this platform takes as input the *μADL* model (arrows numbered 3 in the figure 3) and is performed by the MAY Eclipse plugin (Make Agents Yourself) [28]. MAY involves several phases of generation in which user's intervention is required, for example to choose pre-existing implementations of micro-components. The result of these generations is a platform consisting in a Java archive containing classes (factories, agents type, etc.) to create and support agents whose architectures were defined with *μADL*. Moreover, the resulting library is compliant with the standard JRE. The second aspects aims at providing the behavioural code of agents (arrow number 2 in the figure 3). This generation is achieved thanks to model to text ATL queries. From the decision module of each *AMAS-ML* agent, it generates Java conditional instructions which represent the different situations that the agent have to face and the actions that have to be performed in each situations. This portion of code is contained by a Java class which extends the agent type generated by MAY with the corresponding *μADL* model (see section 3.2).

The consistency between models and code is critical in MDE. In the *INGENIAS* methodology, the MAS designers agree that the best mechanism for the consistency is to keep some programming code at the model. Otherwise, the code would be overwritten each time the code is generated. This programmers' code is inserted in the model by means of a modelling concept called *INGENIAS CodeComponent*. The *CodeUploader* tool performs the reverse engineering. This tool receives the programmers' code that is embedded in the system code, and the tool inserts back this code in the model. The maintenance of models consistency is summarised with the following steps.

1. Developers design models expressed with the *INGENIAS* modelling language, by means of the IDK editor.
2. The *IAF* generates the system code from the models.
3. Developers write code within the systems code.
4. The *CodeUploader* tool uploads the written code in the models.
5. The cycle comes back to first step, if necessary.

In *ADELFE*, a reverse-engineering text-to-model transformation has been defined to maintain the consistency between *μADL* models and the agents of the generated platform. Technically, it uses the Java Development Tool (JDT) Eclipse plugin to navigate Java source code and the *μADL* plugin based on the Eclipse Modeling Framework (EMF) to create the model. The result is a plugin that contains the algorithm written in Java to maintain the consistency between *μADL* models and the platform code.

A key advantage of the model-to-text transformations is that they reduce the coupling between models and their implementation. Whatever are the changes made in the code generation it is still possible to regenerate the code with the same models. For example, migration from a platform to another only implies to modify the transformation, the models do not have to be modified. A drawback is that the consistency between the generated code and the models has to be managed. For this purpose *INGENIAS* use the Code Up-loader and *ADELFE* the reverse engineering. It is also important to notice that enabling source code modification is often a matter of efficiency for developpers. As long as MAS developers are computer scientits, they prefer to describe behaviours or algorithms in programming languages , which therefore require text-to-model loader. However, when MDE users would be business specialists, the odds are that they will prefer to integrate this precise part within the model, even thought it requires to use graphical algorithmic language (like NXT-G graphical language of the LEGO® Mindstorms® NXT system)[6].

4 Challenges for Agent-Oriented Model Driven Engineering

AOMDE faces some challenges that researchers need to address. Some of them are the model transformation by-example and the model upgrade.

4.1 Model Transformation By Example

The application of MTBE to agent-oriented software engineering can improve and fasten the AOMDE. MAS designers are hardly never used to model-transformation languages. Existing agent-oriented tools can be used to specify MAS models from which a MTBE approach can generate model transformations.

In particular, this work proposes that model transformations can be generated from examples in *INGENIAS* modelling. The MTBE approach is specially useful for generating these model transformations, because both source and target model examples can

[6] `http://www.ortop.org/NXT_Tutorial/`

be defined with *INGENIAS*. Figure 4 shows some pairs of models, from which model transformations can be generated for *INGENIAS* methodology. The generated model transformation creates the agent specification (AgentModel diagram) from use cases (UseCase diagrams). As one can observe, this example needs to transform patterns of several modelling elements instead of simple one-to-one transformations.

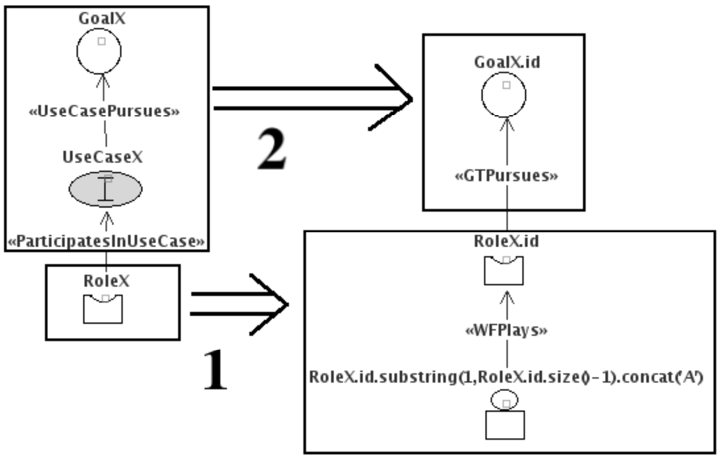
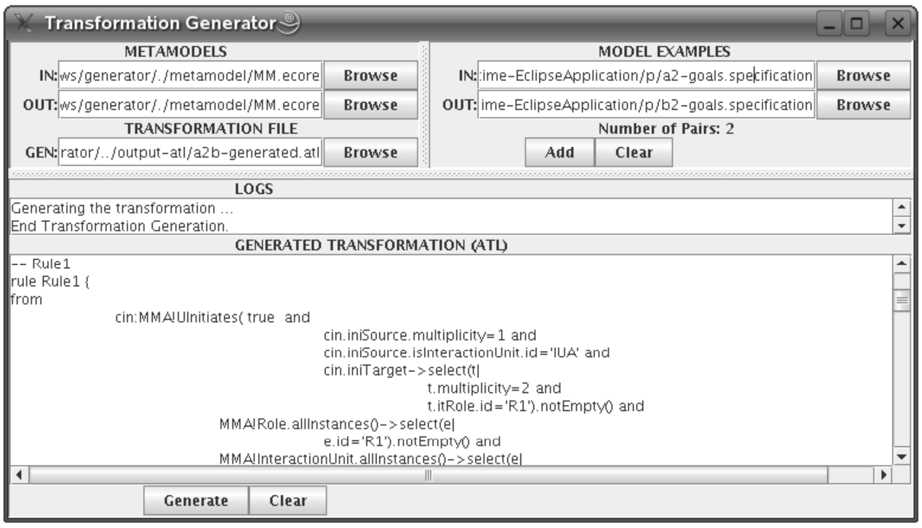


Fig. 4: Model transformations for generating the Specification of Roles. Each square represent a model example for MTBE. Each pair of models is related with an arrow and a number, in which the source model and target model are respectively situated at both sides of the arrow.

Existing MTBE algorithms and tools [32, 33] are only able to generate one-to-one transformations. For this reason, Grasia research group has defined an algorithm for MTBE that overcomes this limitation. This algorithm can generate many-to-many transformations and, in addition, it can create the appropriate mapping of attributes so the generated transformation propagate the information from the source to the target. A prototype tool (see Figure 5) implements this algorithm for generating ATL transformations from *INGENIAS* model, this tool is called *MTGenerators*.

The tool provides an interface (GUI) in which the user can select the input and output meta-models of the transformation. The user must define the meta-models with the ECore language and select the corresponding location paths in the top-left area of the GUI. The user can add the pairs of model with the top-right area of the generator tool, by selecting the corresponding location paths and adding them. After the automatic generation, the tool shows some logs in the *Logs* text area, confirming that the generation has finished successfully. The generated model transformation is shown in the bottom text area of Figure 5. In this manner, the user can examine the generated transformation.

In brief, the presented MTGenerator tool automatically generates a model transformation. Even if the user wants to manually improve the generated transformation, the tool saves time for the user because it provides a generated transformation as a basis for the final model transformation.



IN: ws/generator/./metamodel/MM.ecore
OUT: ws/generator/./metamodel/MM.ecore
GEN: rator/../output-atl/a2b-generated.atl

IN: ime-EclipseApplication/p/a2-goals.specification
OUT: ime-EclipseApplication/p/b2-goals.specification
Number of Pairs: 2

```
Generating the transformation ...
End Transformation Generation.
```

```
-- Rule1
rule Rule1 {
from
            cin:MMA!UInitiates( true  and
                        cin.iniSource.multiplicity=1 and
                        cin.iniSource.isInteractionUnit.id='IUA' and
                        cin.iniTarget->select(t|
                              t.multiplicity=2 and
                              t.itRole.id='R1').notEmpty() and
                  MMA!Role.allInstances()->select(e|
                        e.id='R1').notEmpty() and
                  MMA!InteractionUnit.allInstances()->select(e|
```

Fig. 5: Model-Transformation Generator Tool

4.2 Transformations for Model Upgrades

MAS languages and tools are usually not stable because of continuous evolution. When meta-models of MAS modelling languages change, the existing models need to be upgraded for become conform to the new meta-model. This work proposes to use model transformations to the model upgrade. This section discusses some our experience in model upgrades.

First model upgrade relates to the multiplicity of the agent interactions in *INGENIAS*. IDK 2.6 interactions among agents were only one-to-one. IDK 2.7 supported one-to-many and one to one. In the update, the old interactions added a new attribute indicating their multiplicity. This kind of model update could be manually defined or generated with MTBE.

The second upgrade to introduce the notation for iterative interactions will soon be necessary. Currently, IDK 2.8 iterative interactions are not supported with a specific notation. In the IDK 2.8 models, the *FrameFacts* are used to repeat an interaction. A new IDK version will include soon the specification of iterative interactions without any ad-hoc mechanism. A transformation can achieve the mentioned update detecting a pattern of an iterative interaction and, substituting the mentioned patter with the new representation.

In *INGENIAS*, the kinds of diagram slightly evolves over time. Therefore, the models must change according to the new set of diagram types. The diagrams of the removed types must change to another similar diagram types. These transformations depend on the differences between the old and new sets of diagram types. For instance, if no diagram type is removed then, likely, no transformation is needed. If the diagram types are renamed or changed, a simple transformation is needed. If the diagrams types become more restricted (less entity or relationship types are allowed in a diagram type), then a more complex transformation is needed since some elements of the diagrams must be changed to other diagrams. The transformation must decide which diagram contains the changed elements or create a new diagram if necessary.

5 Related Work

The MDE principles are spreading in the AOSE community. This section presents some works that are particularly representative of this spread. However, it is not intended to present an exhaustive survey of the domain.

Tropos[7] is a agent oriented development process [5] that emphasis the expression of requirements. It is divided into five phases covering the whole life-cycle from the definition of early requirements to the implementation. Tropos is associated with a design tool called TAOM4E (Tool for visual Agent Oriented Modelling for the Eclipse platform) and based on the Eclipse plugins EMF and GEF (Graphical Editing Framework) [4]. TAOM4E uses a specific modelling language which was previously defined in the TAOM tool. Tropos also introduces a model driven approach based on the MDA principles proposed by the OMG (see section 2.1). [27] presents the different types of transformation which were implemented in Tropos for model refinement and translation of preliminary UML2.0 models into Tropos models. TAOM4E offers additional code generation tool[8] for the agent platform JADEX [26]. However, for the moment TAOM4E do not provide means to keep model and code consistent.

Prometheus is a agent oriented methodology [21] which is related to a specific design environment : the Prometheus Design Tool (PDT)[9] [19]. Prometheus covers all phases since the specifications of the system to test or debugging. However, PDT provides means to achieve the three main phases related to system design: the system specification, the architectural design and the detailed design. In the first step designers determine system scenarios, actors interacting with the system in these scenarios, goals of the system, the different roles, their perceptions, their actions, data, etc. The system specification is a preliminary view of the system focussing on goals, roles and scenarios. This first modelling phase is followed by a more precise design that leads to the definition of the agent playing as well as their relationship. Finally, the last step describes the internal structure of the agents in terms of capability, behaviour and protocols in which they play a role. *PDT* allows cross-checking model verification to ensure the consistency of its models (system specification, architectural and detailed design). Moreover, a mapping between the Prometheus models and the JACK platform has been defined [31]. This mapping is implemented in the PDT to generate code [20].

[7] `http://www.troposproject.org/`
[8] `http://se.itc.it/morandini/home.html`
[9] `http://www.cs.rmit.edu.au/agents/pdt/`

MDAD (Model Driven Agent Development) is a development process that applies MDA principles (see section 2.1) for the development of MAS [12]. The PIM is defined thanks to UML based meta-models (profile), covering aspects such as the domain, the agents and the organisation. In *MDAD* the PIM is transformed into a PSM conforming to the INteractive Agent Framework (INAF) meta-model. Jarraya in [12] presents imperative rules to generate the code of the application from the PSM. The generated code is compliant with the INAF architecture.

ASPECS is a model driven methodology based on PASSI and dedicated to the development of complex systems [7]. It is specially dedicated to the design of holonic systems. It proposes a meta-model separated in three related viewpoints, inherited from PASSI, the problem, the agency and the solution domain. The application of the process result in a specific model (PSM) that can be use to generate the code of the application. The code is compliant to the Janus execution platform[10] which shared the same holonic system principles that the solution domain meta-model.

All the aforementioned projects adopt a Model Driven approach to facilitate MAS development. Most of them used model-to-text transformations and some of them define model-to-model transformations to refine PIM into PSM. However, *INGENIAS* and *ADELFE* propose to deal with a wide range of abstraction levels, reverse engineering, re-factoring and are initiating works on MTBE to automate the definition of model transformations. We advocate that these issues that led us to AOMDE should profit to all the MAS methodologies and design tools.

6 Conclusion

Agent-Oriented Model-Driven Engineering (AOMDE) proposes the application of Model-Driven Engineering principles to MAS development. AOMDE is grounded in the following statements. First of all, MAS deals with a wide range of abstraction levels, model transformations can bridge the gap between these levels. Secondly, MAS concepts are continually evolving, meta-models can precisely describe this evolution and support the maintenance of design tool. Finally MAS languages are semantically richer than object oriented languages for instance. Thus, code generation result is larger than the code generate from UML model and the productivity is higher.

Nevertheless, there is still some challenges in AOMDE to cope with. The MTBE is the process of generating model transformations from examples, which is specially relevant for AOMDE since this approach deals with complex transformations between patterns of model elements. The model upgrade is also a mandatory challenge of AOMDE because of the frequent changes in MAS modelling languages.

AOMDE rises specifics needs for which specific solutions are required. These solutions can also profit the whole model-driven community.

[10] `http://www.janus-project.org`

References

1. B. Bauer and J. Odell. Uml 2.0 and agents: how to build agent-based systems with the new uml standard. *Engineering Applications of Artificial Intelligence*, 18(2):141–157, March 2005.
2. F. Bellifemine, A. Poggi, and G. Rimassa. Developing multi-agent systems with a FIPA-compliant agent framework. *Software-Practice and Experience*, 31(2):103–28, 2001.
3. C. Bernon, V. Camps, M.P. Gleizes, and G. Picard. Engineering Adaptive Multi-Agent Systems: The ADELFE Methodology . In Brian Henderson-Sellers and Paolo Giorgini, editors, *Agent-Oriented Methodologies*, volume ISBN 1-59140-581-5, pages 172–202. Idea Group Pub, NY, USA, June 2005.
4. D. Bertolini, L. Delpero, J. Mylopoulos, A. Novikau, A. Orler, L. Penserini, A. Perini, A. Susi, and B. Tomasi. A tropos model-driven development environment. In Nacer Boudjlida, Dong Cheng, and Nicolas Guelfi, editors, *CAiSE Forum*, volume 231 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.
5. P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, May 2004.
6. D. Capera, J.P. Georgé, M.P. Gleizes, and P. Glize. The AMAS Theory for Complex Problem Solving Based on Self-organizing Cooperative Agents . In *TAPOCS 2003 at WETICE 2003, Linz, Austria, 09/06/03-11/06/03*. IEEE CS, June 2003.
7. M. Cossentino, N. Gaud, G. Stéphane, V. Hilaire, and A. Koukam. A holonic metamodel for agent-oriented analysis and design. *Holonic and Multi-Agent Systems for Manufacturing, Lecture Notes in Computer Science*, 4659:237–246, 2007.
8. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, Reading, MA, USA, 1995.
9. J.J. Gómez-Sanz, R. Fuentes, and J. Pavón. Enabling Rapid Prototyping using Decoupling of Code Skeletons and Code generation Process. *INFOCOMP Journal of Computer Science*, February:26–34, 2006.
10. J.J. Gómez-Sanz and J. Pavón. Defining coordination in multi-agent systems within an agent oriented software engineering methodology. *Proceedings of the 2006 ACM symposium on Applied computing*, pages 424–428, 2006.
11. I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.
12. T. Jarraya and Z. Guessoum. Towards a model driven process for multi-agent system. *Multi-Agent Systems and Applications V, Lecture Notes in Computer Science*, 4696:256–265, 2007.
13. S. Kelly. GOPRR Description. *PhD dissertation*, Appendix 1, 1997.
14. S. Leriche and J.P. Arcangeli. Adaptive Autonomous Agent Models for Open Distributed Systems. In *International Multi-Conference on Computing in the Global Information Technology (ICCGI), Guadeloupe, 04/03/2007-09/03/2007*, pages 19–24, http://www.computer.org, March 2007. IEEE Computer Society.
15. S. Leriche and J.P. Arcangeli. AGENTϕ: A Tool for Modeling Composite Self-Adaptive Agents. *International Transactions on Systems Science and Applications*, 4(2):130–138, May 2008.
16. N. Medvidovic and R.N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.
17. Object Management Group. *MDA Guide Version 1.0.1, OMG document ad/2002-04-10 (2002)*, June 2003.

18. Object Management Group. *Unified Modeling Language (UML) 2.0 Superstructure Specification*, OMG edition, August 2003. Final Adopted Specification.
19. L. Padgham, J. Thangarajah, and M. Winikoff. Tool support for agent development using the prometheus methodology. In *Evolvable Hardware*, pages 383–388. IEEE Computer Society, 2005.
20. L. Padgham, J. Thangarajah, and M. Winikoff. AUML protocols and code generation in the Prometheus design tool. *Proceedings of the 6th International Joint Conference on Autonomous Agents and multiagent systems*, pages 1379–1380, 2007.
21. L. Padgham and M. Winikoff. Prometheus: A methodology for developing intelligent agents. *Proceedings of the 3rd International Workshop on Agent Oriented Software Engineering (AOSE 2002), Lecture Notes in Computer Science*, 2585:174–185, 2003.
22. J. Pavon, M. Arroyo, S. Hassan, and C. Sansores. Agent-based modelling and simulation for the analysis of social patterns. *Pattern Recognition Letters*, 29(8):1039–1048, June 2008.
23. J. Pavón and J. Gómez-Sanz. Agent Oriented Software Engineering with INGENIAS. *Multi-Agent Systems and Applications III, Lecture Notes in Computer Science*, 2691:394–403, 2003.
24. J. Pavón, J.J. Gómez-Sanz, A. Fernández-Caballero, and J.J. Valencia-Jiménez. Development of intelligent multisensor surveillance systems with agents. *Robotics and Autonomous Systems*, 55(12):892–903, 2007.
25. J. Pavón, J.J. Gómez-Sanz, and R. Fuentes. Model Driven Development of Multi-Agent Systems. *ECMDA-FA, Proceedings of Model Driven Architecture-Foundations and Applications, Second European Conference, ECMDA-FA 2006, Bilbao, Spain*, 4066:284–298, 2006.
26. L. Penserini, A. Perini, A. Susi, M. Morandini, and J. Mylopoulos. A design framework for generating bdi-agents from goal models. In Edmund H. Durfee, Makoto Yokoo, Michael N. Huhns, and Onn Shehory, editors, *AAMAS*, pages 610–612. IFAAMAS, 2007.
27. P. Perini and A. Susi. Automating model transformations in agent-oriented modelling. *Agent-Oriented Software Engineering VI, Lecture Notes in Computer Science*, 3950:167–178, 2005.
28. S. Rougemaille, J.P. Arcangeli, M.P. Gleizes, and F. Migeon. ADELFE Design, AMAS-ML in Action. In *International Workshop on Engineering Societies in the Agents World (ESAW), Saint-Etienne, 24/09/08-26/09/08*, http://www.springerlink.com/, September 2008. Springer-Verlag.
29. S. Rougemaille, F. Migeon, C. Maurel, and M.P. Gleizes. Model Driven Engineering for Designing Adaptive Multi-Agent Systems. In *The 8th annual International Workshop on Engineering Societies in the Agents World: ESAW*, page (on line), http://www.springerlink.com, October 2007. Springer.
30. J.P. Soto, A. Vizcaíno, J. Portillo, and M. Piattini. Modelling a Knowledge Management System Architecture with INGENIAS Methodology. *Proceedings of the 15th International Conference on Computing*, 2006.
31. J. Sudeikat, L. Braubach, A. Pokahr, and W. Lamersdorf. Evaluation of agent-oriented software methodologies - examination of the gap between modeling and platform. *Agent-Oriented Software Engineering V, Lecture Notes in Computer Science*, 3382:126–141, 2004.
32. D. Varró and Z. Balogh. Automating model transformation by example using inductive logic programming. *Proceedings of the 2007 ACM symposium on Applied computing*, pages 978–984, 2007.
33. M. Wimmer, M. Strommer, H. Kargl, and G. Kramler. Towards Model Transformation Generation By-Example. *Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS 2007*, 40(10):4770, 2007.