Automating Model Transformations in Agent-Oriented Modelling

Anna Perini and Angelo Susi

ITC-irst, Via Sommarive, 18, I-38050 Trento-Povo, Italy
{perini, susi}@itc.it

Abstract. Current Agent-Oriented Software Engineering (AOSE) methodologies adopt a model-based approach for analysis and design, but, in order to become of practical use, they should include it in a clear and customizable software development process and provide CASE tools that support it.

In this regards, the Model-Driven Architecture (MDA) initiative of OMG is providing useful concepts and techniques. The MDA ultimate objective is that of improving quality and software maintainability by allowing for the reuse of models and mappings between models. It offers standards and techniques for model interoperability and for automating model transformations.

Our goal in this paper is to address the role of model transformations in AOSE by discussing a practical example, with reference to the *Tropos* methodology. In particular, we will focus on the automatic transformation of a *Tropos* plan decomposition into a UML 2.0 activity diagram.

We will show how to use the transformation technique to automate model mappings and describe how a CASE tool, based on a modular architecture, has been extended to automate models transformations.

1 Introduction

Modeling techniques are largely used in Agent-Oriented Software Engineering (AOSE). Current methodologies, like Gaia [24], PASSI [7], Prometheus [21], Adelfe [3], *Tropos* [4], propose their own conceptual modeling language and a set of diagrams (or views on the model) to support specific steps in the analysis and design of software. In order to become of practical use the following issues need to be addressed.

First, a model-driven software development process should be clearly defined by specifying the analysis and design steps, with their objectives, set of artifacts to be produced, guidelines and techniques to be exploited to build them.

Second, CASE tools should be provided, at support of the different tasks in model based design such as analysis and verification of models or automatic transformation from one specification language to another, in a transparent and simple manner. These latter aspects are at the core of the Model-Driven Architecture (MDA) initiative of OMG [19].

The ultimate goal of MDA is that of improving the quality of software products and of the development process, by allowing for the reuse of models and the mappings between models. Basically, MDA proposes an approach to software development based on modeling and on the automated mapping of source models to target models. Code

J.P. Müller and F. Zambonelli (Eds.): AOSE 2005, LNCS 3950, pp. 167–178, 2006.

can be seen as a target model as well. So, there is a lot of effort in MDA to develop model interoperability standards, as well as model-to-model transformation concepts and techniques for their automation.

The MDA initiative refers mainly to Object Oriented software development and proved to be effective in relevant application domain, such as web services (business process integration) [18]. Recently, a few proposals to exploit MDA ideas and techniques in Agent Oriented software engineering have been proposed [11, 15, 22].

We think that MDA standards and technological infrastructure are relevant to make AO methodologies usable by practitioners. In particular, adopting MDA standards for model interoperability and for model-to-model automatic transformation could, on one side, support a flexible and customizable software development process, on the other side, offer a complementary approach to the definition of a common metamodel[1].

In this paper we focus on model transformation concepts and techniques in an AO approach to software development, with reference to the *Tropos* methodology. In this methodology the concept of transformation has been introduced also in previous work. Here we will revise and discuss the role of automatic transformations in *Tropos* and describe a tool that supports them.

The paper is structured as follows. Section 2 recalls transformation concepts and techniques in MDA, previous work in *Tropos* and discuss the role of model transformations in *Tropos*. Section 3 and 4 present our approach, focusing on a particular type of transformation in *Tropos* (i.e. synthesis), and present a CASE tool, that supports it. Related works are discussed in Section 5. Finally, conclusion and future work are presented in Section 6.

2 MDA and Model Transformations in *Tropos*

The *Tropos* methodology [4] supports an agent-oriented approach to software development organized in five major phases or disciplines[2]. They are: *Early Requirements*, where a description of the application domain is produced; *Late Requirements*, in which the system-to-be is introduced in the domain and its impact within the environment is analyzed; *Architectural Design* where a representation of the internal architecture of the system is given in terms of subcomponents of the system and relationships among them; *Detailed Design* which focuses on the specification of agents capabilities and interaction; *Implementation*, i.e. the production of code from the detailed design specification, according to the established mapping between the implementation platform constructs and the detailed design notions.

For the first three disciplines *Tropos* adopts a modeling language that allows to represent intentional and social concepts, such as actor and goal, plan, resource, and a set of relationships between them, such as actor dependency, goal or plan decomposition, means-end and contribution relationships. While for the detailed design discipline the use of UML activity diagrams for the agent capabilities specification and of sequence

[1] A currently ongoing effort pursued by the AOSE Technical Forum Group of AgentLink [2].

[2] The term discipline is used according to the definition given in the Unified Process [14], namely a set of activities to be performed in order to produce a particular set of artifacts.

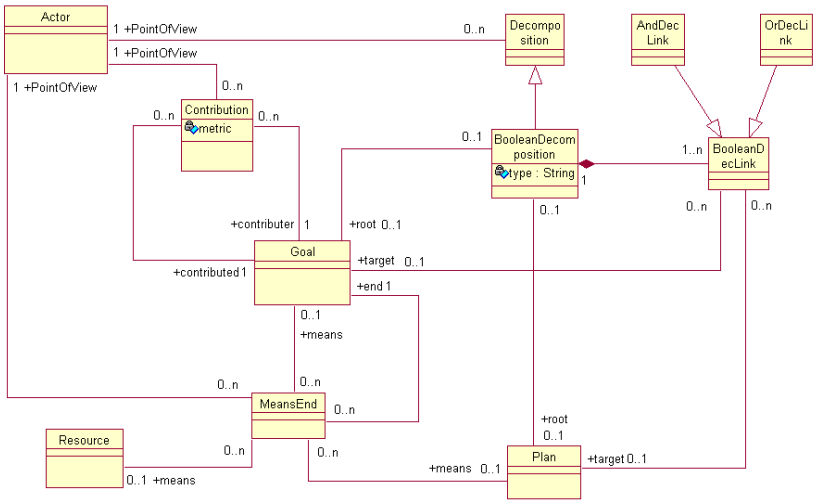

Fig. 1. An excerpt of the *Tropos* metamodel

diagrams for agent interactions specification have been proposed. In [4] a preliminary mapping to the JACK multi-agent platform was defined and applied to a case-study.

Modeling in *Tropos* has been conceived as an incremental process where an initial model is refined by adding new elements and properties by means of the analysis of each actor goals and plans. A description of this process in terms of a non deterministic concurrent algorithm has been given in [4]. Moreover, a first proposal to characterize it in terms of an iterative application of simple transformations has been described in [5].

In the following we will revise the role of transformations in the *Tropos* methodology in the light of the MDA framework. We will first recall the basic goals and concepts of MDA, then discuss how they can be adopted in CASE tools for supporting *Tropos*.

MDA considers models as corporate assets which can evolve independently of the relative code. Models can be partially reused or mixed with other models to generate a new system [19]. Models can be specified from different views and can be represented at different levels of abstractions.

Concerning model transformation, the basic idea proposed in MDA is that of defining the meta-models of source and target modeling languages according to a standard and to define mapping and transformation mechanisms between meta-model elements. The transformation of a source model into a target model will derive in a straightforward way from the transformation mechanisms defined at the meta-model level, since the models are instances of the correspondent language metamodel.

The MDA's meta-modeling standard is the Meta Object Facility (MOF) [16] which defines a set of modeling construct that allow to manage meta-models interoperability. For instance, it offers a standard mechanisms for automatically deriving a concrete syntax based on XML DTDs and/or schemas known as XML Model Interchange (XMI). An example of MOF compliant meta-model is illustrated in Figure 1 which depicts an excerpt of the *Tropos* modeling language metamodel.

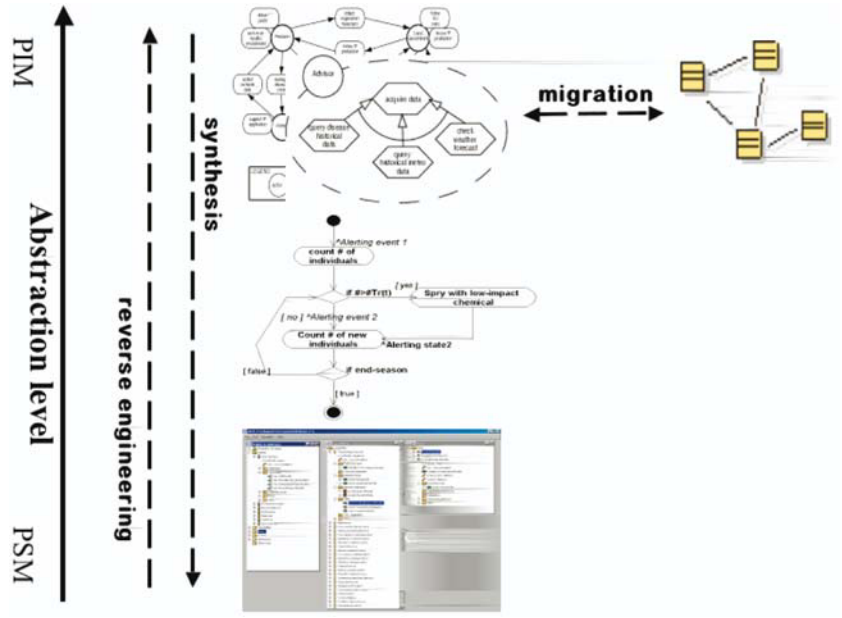

Fig. 2. Model Translation in MDA: an adaptation of the schema proposed in [18] to the *Tropos* methodology. The model abstraction level increases from Platform Specific Model (PSM), represented by JACK code, to Platform Independent Model (PIM), represented by UML and *Tropos* models.

A language for describing the generic transformation of any well formed model is not yet available in a standard form. A first step in the standardization process has been performed by OMG by issuing a request for proposal on Query/View/Transformation (MOF QVT [12]) which should take into account requirements such as that of defining a language for querying MOF models; giving a language for transformation definitions; allowing for the creation of views of a model. Several techniques for model transformation have already been proposed.

The role of transformations in *Tropos* can be discussed referring to a classification of QVT model transformations that have been proposed in [18], which uses the terminology introduced by Visser for program transformation [23]. Language *translation*, and language *rephrasing* are top level processes. Basically, in the former, a model is transformed into a model of a different language, and in the latter, a model is changed, in some way, into a same language model.

Figure 2 depicts the different translation processes in MDA, according to this classification. *Migration* is a type of translation in which a model is transformed to another one, or to a language dialect, at the same level of abstraction. For instance, if we intend to integrate *Tropos* architectural design with UML design we may need to migrate from actor diagrams to package / class diagrams. Another example of this type of transformation occurs when we need to specify behavioral properties of a model by temporal logic annotation (*FT*). An automatic transformation mechanism, from informal *Tropos* to *FT*, has been built adopting a visitor-based approach, as described in [22]. *Synthesis*

is a type of translation in which a model is transformed to another one at a lower level of abstraction. This type of transformation in *Tropos* occurs when building the detail design model from the architectural design model, that is when we need to add specification of agent capabilities and of agent interactions. In this paper we will focus on this example considering, in particular how an actor (agent) plan decomposition can be automatically translated into a capability diagram (UML 2.0 activity diagram). *Reverse engineering* is the inverse transformation.

Rephrasing refers to different transformations that may occur when building and refining a model; *normalization* consists in a transformation of a model by reducing it to a sub-language; *refactoring*, concerns restructuring a model with the objective to improve it; *correction*, i.e. fixing possible model errors; and *adaptation* of a model in order to bring it up to date with new or modified requirements. The previously cited work [5] on defining the modeling process in terms of an incremental application of basic transformations was intended to support this type of transformation processes. Moreover, a first proposal of applying graph transformation techniques to its automation is described in [20].

We are currently interested in exploring the problem of transformation between two modeling languages defined by different metamodels, and in particular in maintaining the synchronization between the models. This is required in the *Tropos* methodology when we deal with the transition from a *Tropos* Architectural Design model, to a Detailed Design specification. Notice that the Architectural Design model is specified according to the *Tropos* metamodel as defined in [4], while for the second (which includes UML activity diagram, sequence diagram) we aim at exploiting the UML 2.0 metamodel and at maintaining the traceability between the models.

3 Automating *Tropos*-to-UML Model Transformation: An Example

Among the different approaches for model-to-model transformations that have been recently proposed, we focused on two of them namely: the Graph Transformation (GT) [13] and a Frame Logics [17]. In [20] we describe how to apply GT to *Tropos* model rephrasing transformation. Briefly GT approach is based on set of rules that represents the status of a certain sub-graph of the models before and after the application of the rule. In particular these rule's sub-graphs can be related respectively to the source and target metamodel. Some problem arises when we deal with GT specifications. In fact this framework introduces non determinism in at least two phases: in order to apply a rule we have first to choose it, and then we have to choose the sub-graph of the source model in which the rule has to be applied. The result of the transformation strictly depends on these choices. Some restrictions can be adopted in order to reduce this phenomenon: the next rule to be applied can be chosen on the basis of the rules applied before, or the application of the rules can be executed on the basis of a priority list. Another possible problem is the possibility to assure the termination of a sequence of rule application. Also in this case some hypothesis can be made in order to limit the problem.

We are exploiting a Frame Logics based approach described in [6] to deal with Metamodel transformation between the *Tropos* and the UML 2.0.

In particular this approach is based on the definition of some properties of the target model in terms of the source model, avoiding the specification of the process used to obtain the target, and it takes into account the mandatory requirements of the MOF QVT consortium related to the Query/View/Transformation framework. In particular the proposal defines a language for querying MOF-compliant models (or set of models) and a subset of this language can be used to specify transformation of MOF-compliant models. The transformations can be automated and views of models can be obtained via transformations. This approach leads to a simpler semantic model, respect, for example, to the GT techniques; this made easier the understanding of the transformation rule. Moreover it does not need any hypothesis related to the ordering in which the rules have to be applied or to the termination of the transformation.

The transformation language proposed in the approach consists of three major concepts: *pattern definitions*, *transformation rules*, *tracking relationships*. *Pattern definitions* are generated in order to identify structures that are used several times in a given transformation. *Transformation rules* allow to specify the target configuration in terms of the entities in the source configuration. *Tracking relationships* are used to associate the target elements with the source elements that lead to their creation allowing to maintain the traceability between source and target model instances entities. Moreover the work proposes a syntax for the rules composed by some clauses; some of them (e.g. the *Forall* and *Where*) are used by the rule to recognize some pattern in the instance of the source model, while other (e.g. *Make* and *Set*) are used to build the instance of the target model.

We will show how we applied it in *Tropos* showing an example of transformation from *Tropos* plan decomposition structure to a UML 2.0 Activity Diagrams.

A *Tropos* plan decomposition represents a graph describing a hierarchical relationship between the root plan and the sub-plans. Let us consider the case of an AND plan decomposition as the one represented in Figure 3 a).

Fig. 3. A *Tropos* plan decomposition diagram for a given Actor and the corresponding UML 2.0 activity diagram

The meaning of the decomposition is: the root Plan A can be decomposed in the sub-plans Plan B and Plan C; both of them have to be executed in order to have the root plan executed. This hierarchy identifies a set of possible plans composed by the set of sub-plans. In particular nothing is specified about the order in which the set of sub-plans have to be executed.

```
TRANSFORMATION Tropos2UML: Tropos -> uml2

RULE PlanNoDec2Activity()
  FORALL Plan c
  WHERE NOT (c.booleanDecomposition=BooleanDecomposition)
      AND NOT (c.boolDecLink=BooleanDecLink)
  MAKE Action f, InitialNode Initial, Final Node Final, ControlFlow ToA, ControlFlow ToFin
  SET f.name="noDec", ToA.source=Initial, ToA.target=f, ToFin.target=Final, ToFin.source=f;

CLASS ActionForPlanDec {
  Plan pln;
  Action act;};

RULE PlanDec2Action(c,a,join,fork)
  FORALL Plan c
  WHERE Root(c)
  MAKE Action a, JoinNode join, ForkNode fork, InitialNode Initial,
      FinalNode Final, ControlFlow initToA, ControlFlow AToFin,
      ControlFlow AToFork, ControlFlow JoinToA
  SET a.name=c.name, a.redefinedElement=join, a.redefinedElement=fork,
      ......
LINKING ActionForPlanDec WITH act=a, pln=c;

RULE SubPlan(c,a,join,fork,d,b)
  EXTENDS PlanDec2Action(c,a,join,fork)
  FORALL Plan d
  WHERE ActionForPlanDec LINKS pln=c
      c=d.boolDecLink.BooleanDecomposition.rootPlan
  MAKE Action b, ControlFlow ForkToB, ControlFlow bToJoin
  SET b.name=d.name, a redefinedElement=b, ForkToB.name="ForkToB",
      ...

PATTERN Root(c)
  WHERE c.booleanDecomposition.type="and";
```

Fig. 4. The transformation specification defined in the grammar described in [6]

The plans in the *Tropos* plan diagram are translated into action nodes in the UML activity diagram; moreover from the structure of the plan decomposition it is possible to derive a basic structure for the resulting activity diagram.

In particular the assumption is that the Plan A can be mapped into an activity node, containing a structure composed by the activities corresponding to the plans B and C; moreover in the example the assumption is that the two plans has to be executed in parallel since no information is given about the sequence of the plans in the *Tropos* plan diagram. Figure 3 b) shows the resulting activity diagram.

The transformation shown in Figure 4 is specified via a subset of the grammar described in [6]. In the transformation definition it is possible to distinguish Rules and Pattern used to specify in a declarative way the transformation. The RULE *PlanNoDec2Activity* is for the transformation of the plan decomposition leaves, not decomposed, to an activity in the UML activity diagram. The role is composed by clauses. In the *PlanNoDec2Activity* rule, the clauses FORALL and WHERE retrieve the set of plans that are not decomposed; the clauses MAKE and SET are in charge to build the structure of the corresponding activity diagram, creating a new activity for every retrieved plan, and the links to other activities and control flow components in the

diagram. The RULE *PlanDec2Activity* refers to decomposed plans and transforms them into UML actions that can then be further decomposed in other actions and control structures. In particular in our case fork and join control structures are added together with the action derived from the hierarchy root plan A. The RULE *SubPlan* redefine the rule for the decomposable actions in order to incrementally add new sub-actions in the activity diagram.

In the example the directive PATTERN recognizes the kind of decomposition the transformation has to face with; in this case the pattern recognizes the root of an "and" decomposition, a typical structure in the *Tropos* plan decomposition diagram.

For the sake of clearness, we described the simplest case of a plan and-decomposition structure. Typical cases require to deal with plan or-decomposition or temporal relationships [22] between sub-plans as the one shown in Figure 5. In this case a few additional rules can be defined within a limited effort.

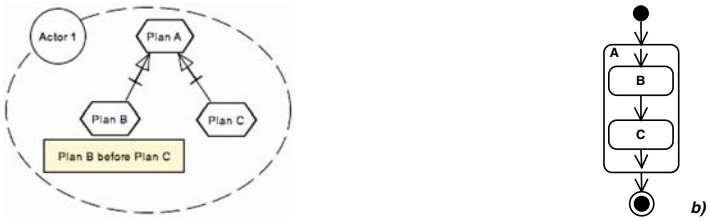

Fig. 5. A *Tropos* plan AND-decomposition diagram with temporal annotation for a given Actor and the corresponding UML 2.0 activity diagram

As described above a relevant issue for us is the possibility of having the synchronization between models and the reversibility of a transformation. The declarative transformations approach shown in [6] partially supports synchronization and reversibility in an automatic way. In general the reverse transformations has to be explicitly defined.

4 A CASE Tool

In this section we focus on the description of a set of tools for supporting the use of the *Tropos* methodology according to the MDA perspective. This requires, first to adopt MOF compliant modeling tools (i.e. whose respective modeling languages' metamodels are specified according to the MOF standard), second, to define model transformations in terms of mapping between the metamodels of the source and the target specification languages.

For instance, a CASE tool at support of the *Tropos* process discussed in the previous section should allow the analyst to build a *Tropos* model (in our case a plan decomposition diagram) using a modeler which includes the *Tropos* metamodel. Part of the model should be automatically translated into a UML model which should be editable by a UML modeler (which includes the UML metamodel). Modifications performed on the UML model should be automatically reflected into the *Tropos* model.

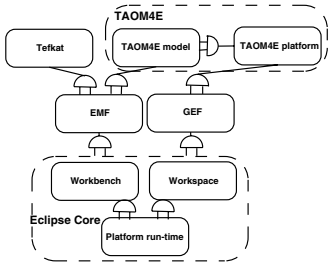

Fig. 6. The architecture of TAOM4e

A *Tropos* modeler called TAOM compliant with MDA metamodel interoperability standards has been described in [22]. The need of a higher flexible architecture which allow to easily extend it induced us to consider the opportunity to re-engineering this tool in the Eclipse Platform [1] that offers a flexible and (economically) convenient solution to the problem of component integration. The Eclipse Platform is an open source initiative that offers a "reusable and extensible framework for creating IDE-oriented tools" [10]. New tools are integrated into the platform through plug-ins that provide new functionalities to the environment. A plug-in is the smallest unit of function in Eclipse. The Eclipse Platform itself is organized as a set of subsystems (implemented in one or more plug-ins) built on the top of a small runtime engine, as depicted in Figure 6. Plug-ins define extension points for adding behaviors to the platform, that is a public declaration of the plug-in extensibility. More precisely, a "plug-in manifest" file specifies the extensions it uses and the extension points it defines.

Figure 6 depicts the architecture of the new modeler (called TAOM4e) and of how it has been extended with a model transformation plug-in. In particular, TAOM4e has been built on top of two existing plug-ins. First, the Graphical Editing Framework (GEF) plug-in[3] that allows developers to create a rich graphical editor from an existing application model. The functionality of the GEF plug-in helps covering one of the most essential requirements of the modeler, that is supporting visual development of Tropos model by providing some standard features like drag & drop, undo-redo, copy & paste and other.

Second, the EMF plug-in[4] which offers a modeling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model. Most important of all, EMF provides the foundation for interoperability with other EMF-based tools and applications.

The TAOM4e component consists of two plug-ins, as depicted in Figure 6, namely, the *TAOM4e model* which implements the Tropos meta-model extending the EMF plug-in and the *TAOM4e platform* which implement the modeler functions needed for building and managing a Tropos Model. It extends the GEF plug-in and the *TAOM4e model* plug-in.

[3] http://www.eclipse.org/gef/
[4] http://www.eclipse.org/emf/

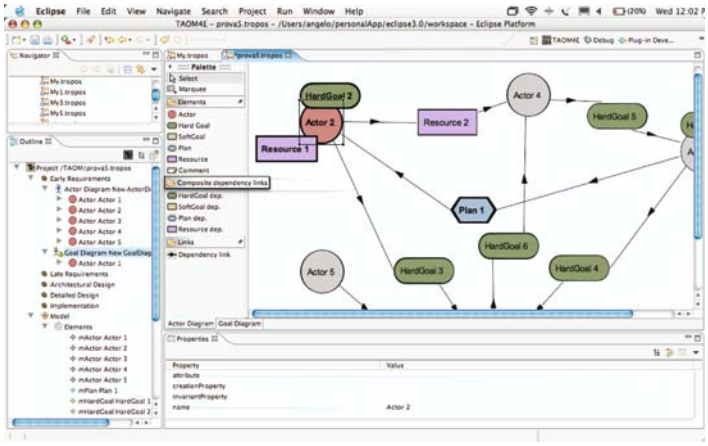

Fig. 7. A snapshot of the TAOM4e Graphic User Interface

The transformation plug-in we used is a model transformation engine called Tefkat[5] that implements a subset of the requirements and of the transformation language described in the DSTC proposal for MOF QVT as in [6]. The tool consists of a set of Eclipse plug-ins based on EMF. Tefkat allow to specify transformations between metamodels that are specified via an XMI compliant notation used by the EMF for the definition of metamodels. Moreover, due to the plug-in structure, Tefkat can easily interact with other Eclipse plug-ins devoted to model definition and management.

5 Related Work

Several works related to Agent-Oriented Software Engineering dealt with the concept of transformation as already pointed out in the first two sections [3, 7, 21]. In particular in [5] a transformational approach to support the analyst during the *Tropos* software development process has been proposed and in [20] Graph Transformation were applied to support the analyst in choosing and/or validating possible model refinement actions. A first proposal to use Graph Transformation in AOSE is given in [11] where this technique is adopted both to capture agent-specific aspects and to define a formal semantics in the definition of an agent-oriented modeling technique. In [15] a work which applies the MDA idea of transform a Platform Independent Model to a Platform Specific Model is proposed; in that case the Platform Specific Model refers to the JACK platform.

More generally QVT proposals are of particular interest for our work. An interesting classification of them can be found in [9]. Moreover interesting ideas on how to apply the MDA framework to a specific domain is given in [18]. In this work automatic transformations between source and target models are proposed in the case of busi-

[5] Tefkat is part of Pegamento project of the DSTC in the University of Queensland http://www.dstc.edu.au/Research/Projects/Pegamento/tefkat/index.html

ness process integration, when dealing with the complexity of large business processes mapping from visual languages to code.

6 Conclusion and Future Work

In this paper we focused on the role of model transformations in an agent-oriented software development by adopting concepts and techniques that are proposed in the MDA initiative by OMG [19].

MDA offers a meta-modeling standard, the Meta Object Facility (MOF) [16], which allows model and meta-model interoperability and is managing the standardization process of model transformations which should be compliant with the so called Query/View/Transformation (MOF QVT [12]) requirements. Several techniques have been already proposed. Although MDA refers mainly to Object Oriented software development its concepts and techniques may be adopted Agent Oriented software engineering as well [8, 11, 15, 22].

In particular, in Section 2 we considered different types of model transformations that can support software development in the *Tropos* methodology and revised how the concept of transformations have been addressed in previous works. We think that most of the considerations can be applied also to other AOSE methodologies.

We presented a (simple) practical example concerning the automatic transformation of a *Tropos* plan decomposition into a UML 2.0 activity diagram (a transformation type called *synthesis* in Section 2), by adopting a declarative transformation language proposed in [6] and we pointed out critical issues such as model synchronization. This type of transformation supports the transition between architectural design and detailed design in *Tropos*, but we may consider to adopt it also for supporting translation between *Tropos* models and UML models referring to a same level abstraction (for instance during architectural design).

We showed also how we are extending a CASE tool implemented in the ECLIPSE platform which offers a highly modular and flexible architecture, to include automatic model transformations.

References

1. *ECLIPSE Platform Technical Overview*, object technology international edition, July 2001. http://www.eclipse.org.
2. C. Bernon, M. Cossentino, M. P. Gleizes, P. Turci, and F. Zambonelli. A Study of Some Multi-agent Meta-models. In *Agent-Oriented Software Engineering V: 5th International Workshop, AOSE 2004*, volume 3382 of *Lecture Notes in Computer Science*, pages 62 – 77, New York, USA, NY, July 2004.
3. C. Bernon, M. Gleizes, S. Peyruqueou, and G. Picard. ADELFE, a Methodology for Adaptive Multi-Agent Systems Engineering. In *Third International Workshop Engineering Societies in the Agents World (ESAW-2002)*, Madrid, Spain, 2002.
4. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, July 2004.

5. P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Modeling early requirements in Tropos: a transformation based approach. In M. Wooldridge, P. Ciancarini, and G. Weiss, editors, *Agent-Oriented Software Engineering II*, volume 2222 of *LNCS*, pages 151–168. Springer-Verlag, 2001.
6. CBOP, DSTC, and IBM. MOF Query/Views/Transformations, 2nd Revised Submission. Technical report, 2004.
7. M. Cossentino. Different perspectives in designing multi-agent systems. In *Proc. of AGES '02*, Erfurt, Germany, 2002.
8. M. Cossentino, P. Burrafato, S. Lombardo, and L. Sabatucci. Introducing Pattern Reuse in the Design of Multi-agent Systems. In *Agent Technologies, Infrastructures, Tools, and Applications for E-Services 2002*, pages 107 – 120, 2002.
9. K. Czarnecki and S. Halsen. Classification of Model Transformation Approaches. In *OOPSLA'03 Worshop on Generative in Context of Model-Driven Architecture*, 2003.
10. J. D'Anjou, S. Fairbrother, D. Kehn, J. Kellerman, and P. McCarty. *The Java developers guide to Eclipse*. Addison-Wesley, 2004.
11. R. Depke, R. Heckel, and J. M. Küster. Agent-Oriented Modeling with Graph Transformation. In P. Ciancarini and M. Wooldridge, editors, *AOSE*, volume 1957 of *Lecture Notes in Computer Science*, pages 150 – 120. Springer, 2001.
12. T. Gardner, C. Griffin, J. Koehler, and R. Hauser. A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standar. In *MetaModelling for MDA Workshop*, York, England, 2003.
13. S. Gyapay and D. Varró. Automatic Algorithm Generation for Visual Control Structure. Technical report, Dept. of Measurement and Information System, Budapest University of Technology and Economics, Hungary, 2000.
14. I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.
15. G. B. Jayatilleke, L. Padgham, and M. Winikoff. Towards a Component-Based Development Framework for Agents. In G. Lindemann, J. Denzinger, I. Timm, and R. Unland, editors, *Multiagent System Technologies, Proceedings of the Second German Conference, MATES 2004*, number 3187 in LNAI, pages 183 – 197. Springer-Verlag, 2004.
16. S. R. Judson, R. B. France, and D. L. Carver. Specifying Model Transformations at the Metamodel Level, 2004. http://www.omg.org.
17. M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of ACM*, 42(4):741 – 843, 1995.
18. J. Koehler, R. Hauser, S. Sendall, and M. Wahler. Declarative techniques for model-driven business process integration. *IBM Systems Journal*, 44(1), 2005.
19. S. J. Mellor, K. Scott, A. Uhl, and D. Weise. *MDA Distilled*. Addison-Wesley, 2004.
20. A. Novikau, A. Perini, and M. Pistore. Graph Rewriting for Agent Oriented Visual Modeling. In *In Proc. of the International Workshop on Graph Transformation and Visual Modeling Techniques, in ETAPS 2004 Conference*, Barcelona, Spain, 2004.
21. L. Padgham and M. Winikoff. Prometheus: a methodology for developing intelligent agents. In *AAMAS*, pages 37–38, 2002.
22. A. Perini and A. Susi. Developing Tools for Agent-Oriented Visual Modeling. In G. Lindemann, J. Denzinger, I. Timm, and R. Unland, editors, *Multiagent System Technologies, Proceedings of the Second German Conference, MATES 2004*, number 3187 in LNAI, pages 169–182. Springer-Verlag, 2004.
23. E. Visser. A survey of strategies in program transformation systems. *Electr. Notes Theor. Comput. Sci.*, 57, 2001.
24. F. Zambonelli, N. R. Jennings, and M. Wooldridge. Developing Multiagent Systems: The Gaia Methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317 – 370, July 2003.