ITC irst

CENTRO PER LA RICERCA
SCIENTIFICA E TECNOLOGICA

38050 Povo (Trento), Italy
Tel.: +39 0461 314312
Fax: +39 0461 302040
e−mail: prdoc@itc.it − url: http://www.itc.it

Model Driven Architecture approach in Tropos

Novikau A.

June 2004

Technical Report # T04−06−03

Model Driven Architecture approach in Tropos

Aliaksei Novikau

PhD student, International Graduate School in Information and Communication Technologies - XVIII cycle, University of Trento, Italy with a fellowship from ITC-IRST, Trento, Italy

Abstract

The OMGs Model Driven Architecture (MDA) is a software development approach which considers models as the primary artifacts. Visual modelling together with model transformations play the key roles in MDA. MDA concentrates on later phases of software development process like architecture and detailed design. Hoverer we think that the idea can be generalized also for other phases like requirements analysis. *Tropos* is the AO approach to engineering distributed systems adopting Multi-Agent paradigm, the analysis of requirements is one of the essentials of *Tropos*. Visual modelling is also the key activity in *Tropos* and this conception actually shares basic motivations with MDA. In this paper we are reviewing how our approach of Graph Rewriting (GR) for *Tropos* visual modelling coincides with MDA ideology.

Key words: Model Driven Architecture, visual modeling, graph rewriting, agent-oriented modeling, Tropos

1 Introduction

The OMGs Model Driven Architecture (MDA) is a software development approach in which models are the primary artifacts [1]. MDA considers the software development process as a sequence of models starting from more abstract and finishing with more detailed and specific. Visual modelling together with model transformations play the key roles in MDA. That is why MDA should be supported by automated tools and services for both defining the models and facilitating transformations.

Model Driven Architecture concentrates on later phases of software development process like architecture and detailed design and on Object Oriented paradigm. Hoverer we think that the idea can be generalized also for other paradigms like Agent Oriented (AO) software engineering and other phases like requirements analysis.

Tropos is the AO approach to engineering distributed systems adopting Multi-Agent paradigm, the analysis of requirements is one of the essentials of *Tropos* [2]. Visual modelling is also the key activity in *Tropos* and this conception actually shares basic motivations with MDA. In the paper we are reviewing how our approach of Graph Rewriting (GR) for visual modelling in *Tropos* [3] is compliant with MDA ideology. From this perspective, we suppose two activities for *Tropos* visual modeling that should be supported with automated tools:

- different views of the *Tropos* model should be conforming between each other;
- models of different phases of the modeling process should be consistent between each other;

The paper is structured as follows: in Sec. 2.1 we will give a brief description of the MDA approach. Some essentials from GR theory are pointed out in Sec. 2.2. Sec. 2.3 presents the *Tropos* modeling language. Sec. 3 describes our vision of MDA and *Tropos* compliancy and then our approach of GR for visual modeling in *Tropos*, namely, the set of graph rewriting rules for building a *Tropos* model; a set of rules for supporting instance creation; derivation transformation technique and how it can support consistency between models of different phases. Finally, conclusions and future work are sketched in Sec. 4.

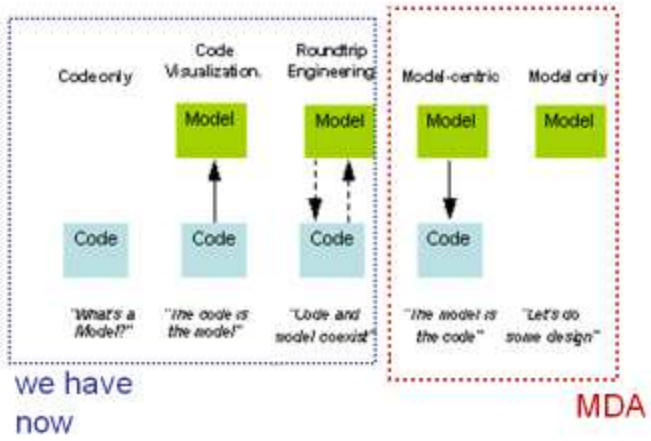

Fig. 1. Different types of software development.

Fig. 2. The vision of IBM on MDA.

2 Background

2.1 Model Driven Architecture

As defined by the Object Management Group (OMG), MDA is a way to organize and manage enterprize architectures supported by automated tools and services for both defining the models and facilitating transformations between different model types [1].

There are different types of software development depicted in Fig. 1. Today, a majority of software developers still prefer the code-only approach (left end of the modeling spectrum, Fig. 1) and do not construct any models at all. This results in real difficulties if the code should be changed or maintained. That is why some developers use reverse engineering tools and obtain the "visualization model" for better understanding the insides of the code. More advanced technique is the "round trip engineering", which offers the bi-directional exchange between the model describing the system architecture (or design), and the code. The developer typically elaborates the system design to some level of detail, then creates a first-pass implementation by applying model-to-code transformations, usually manually. These manual transformations are unreliable and error prone. Therefore the design and implementation team should be very accurate supporting software engineering process of this kind. Usually, after a number of iterations, the difference between the implementation and the model becomes to be so critical that the model and the code can not be synchronized anymore.

MDA approach considers models as the main artifact of software development process. Model allows us to easily implement changes (i.e to support *maintainability*). Also we can check some properties of the model before implementation and avoid very expensive changes of the code later (i.e. *simulations, model checking*). We can explore models to extract a parts of the system that can be reused for constructing another software of the same kind (i.e *reusability*). Therefore MDA concentrates on the last two ways for software development depicted in Fig. 1. Here model plays the centric role and the code is considered as one of

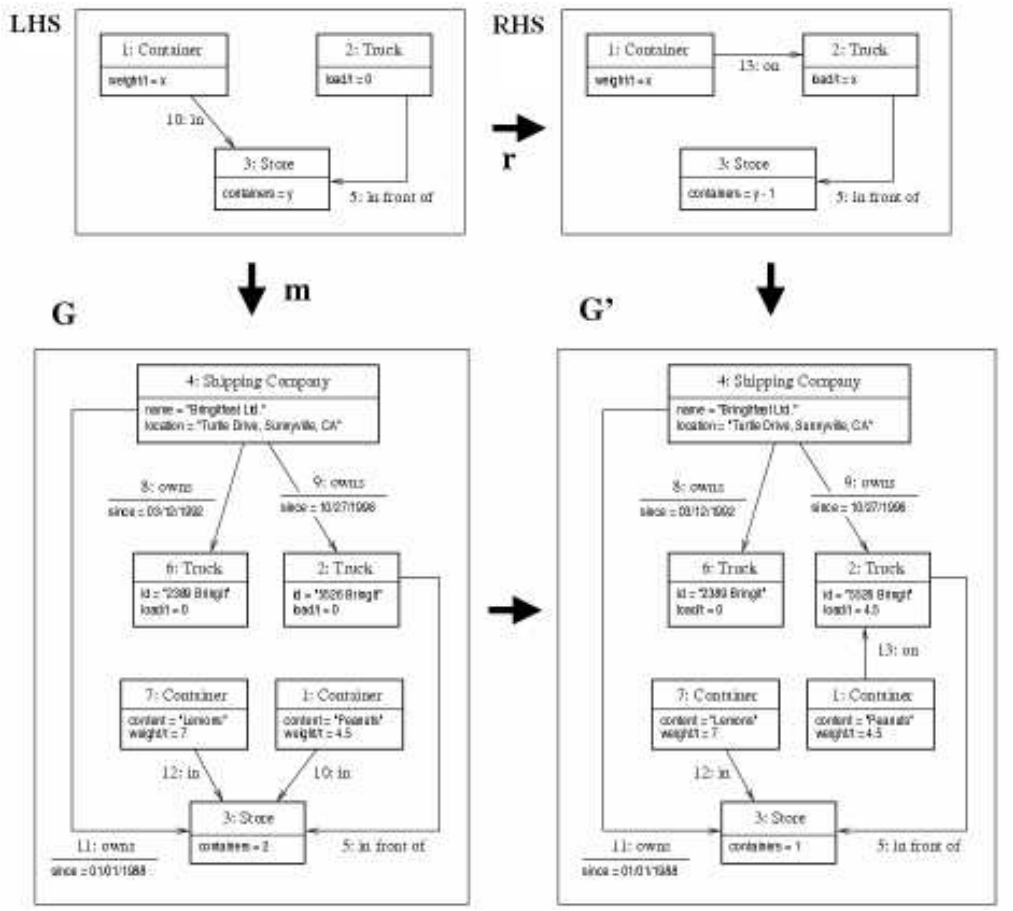

Fig. 3. An example of the graph rewriting rule applied to the UML diagram. The rule moves the *Container* from the *Store* to the *Truck* (removing the the *in* link from the *Container* to the *Store* and creating the *on* link from the *Container* to the *Track*). It also decrements the number of containers in the *Store*, and adds the weight of the *Container* to the *Track*. The rule is applied to the host graph and moves the *Container 1*.

the models. The rightmost case in Fig. 1 considers the option where the model is used for understanding of some essential properties or for simulation of the system (it might not have an implementation). That is why IBM sees MDA approach as the highest abstraction level of creation software as it is shown in Fig. 2.

Visual modelling and transformations between models are accepted as the main two concepts of MDA approach. The process of software development is foreseen as a chain of models beginning from abstract and finishing with more detailed and specific models, including the implementation. The process should be supported with a set of CASE tools. While visual modeling is quite for a long time in practice of SE (the example can be UML [4]), the transformations between different models are made by hands and thus unreliable and error prone. Consequently, OMG considered transformations as the very important part of MDA and announced Request for Proposals for Query/Transformation/View language. The language is going to be used for definition of model transformations. There are several proposals for QVT and some of them use Graph Rewriting as the paradigm for transformations. The examples can be UMLX [5] and GREAT approaches [6].

Since our approach of visual modeling in Tropos is also based on Graph Rewriting techniques [3], the essentials of GR are going to be reviewed in the next subsection.

2.2 Graph Rewriting

The main idea of GR [7] was to generalize the concatenation of strings to a gluing construction for graphs. A graph rewriting is usually defined in terms of a set of *production rules*. A production rule consists of a *left-hand side (LHS)* graph and *right-hand side (RHS)* graph (an example is depicted in Fig. 3). If the LHS graph matches a part of a given graph (called the

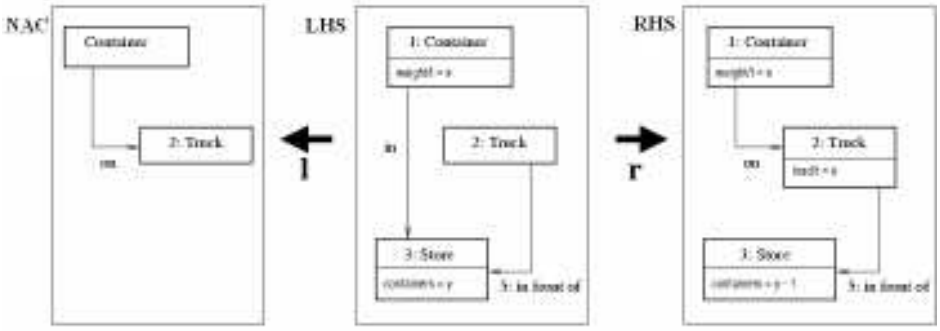

Fig. 4. This NAC assures that the *Track* to be loaded does not already contain a *Container*, there are can be many NACs for one rule.

host graph), then that part of the host graph can be transformed into the RHS graph.

The overlapping objects in the LHS graph and the RHS graph are called the *context* of the rule (i.e. the classes *Container*, *Track* and *Store* and the relation *in front of* in Fig. 3). These nodes and edges remain unchanged during a transformation and function as central points where the transformation takes place. The LHS graph objects not included in the context are deleted while the rule is applied (see the edge *in* of LHS in Fig. 3) and the RHS graph objects are added (glued) to the context (see the edge *on* of RHS in Fig. 3). An application of a GR rule is called a *derivation*, and a sequence of rules applications that transformed an initial graph G_0 into a final graph G_n is a *derivation history*.

There are several algebraic approaches for GR. Usually the approaches and their properties are defined in *Category Theory*. This means a high level of abstraction and thus, the possibility to adopt the results for different graph structures [7]. *Typed attributed graph rewriting* is one of the quite widely used structures. Such GR grammars manipulate graphs where nodes and edges have types from a typeset. Also they are attached with strings, numbers or elements of an algebra. In parallel to a GR rule application on the graphical part, some operations on the attribute part can be performed. Graph grammars with a set of application conditions like a *precondition* (which is represented by the LHS), *Negative Application Conditions or NACs* (additional graphs that should not be met while the LHS mapping to a host graph, in Fig. 4 there is the example of NAC for the rule depicted in Fig. 3) and *conditions on attributes* are widely used in the area of visual modeling.

There are several tools supporting Graph Rewriting techniques. Algebraic Graph Grammar (AGG) [8] can be considered one of general purpose tools for GR. It provides an algebraic GR approach and can manipulate typed attributed graph grammars. A rule can have NACs and conditions on attributes. AGG graph attributes can be annotated with Java code which means that Java classes and standard Java libraries can be exploited to compute attributes while rewritings. The tool environment provides editors for specifying graphs, rules, and the associated Java code, allowing users to play what-if simulations, by specifying the starting graph and the order rules that should be applied. Moreover, being implemented in Java, AGG, can be used as a core graph transformation engine for all those Java applications that need to manage and transform graphs.

2.3 Tropos Agent-Oriented methodology

Tropos is an AO methodology for building distributed software systems which provide a visual modeling language, based on the agent paradigm [2]. The methodology considers five main development phases: early requirements, late requirements, architectural design, detailed design, and implementation. Visual modeling is used for the analysis of the application domain (early requirements phase) as well as for the system requirements (late requirements phase), and for the architectural and the detailed design of the system-to-be. Among the basic concepts provided by the language: the concept of an *actor* for modeling entities which have strategic goals and intentionality, such as a physical agent, a role in an organization, or a component in the system-to-be; the concept of a *goal* for representing the strategic interests of an actor; the concept of a *dependency* between two actors which indicates that an actor depends on another in order to achieve a goal, execute a plan, or exploit a resource. The

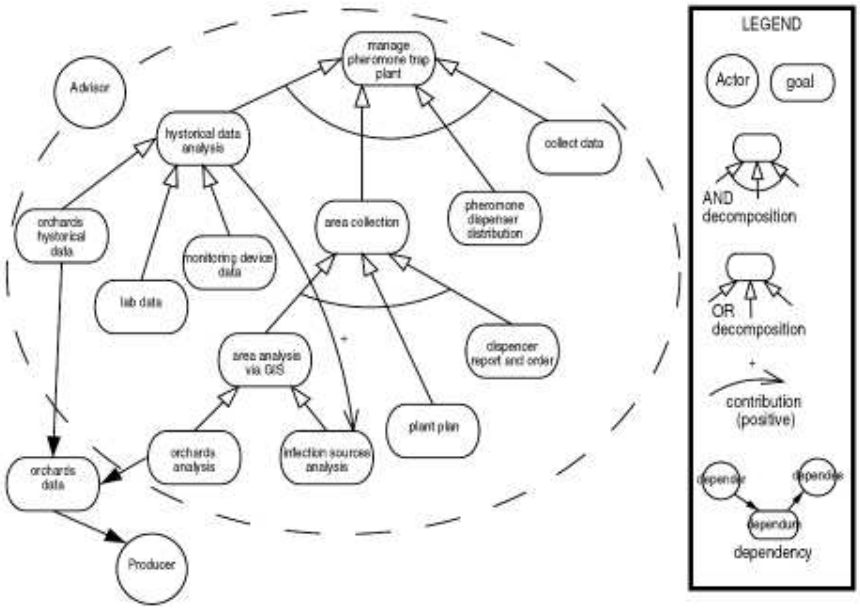

Fig. 5. An example of a *Tropos* actor and goal diagram. The example is taken from [9].

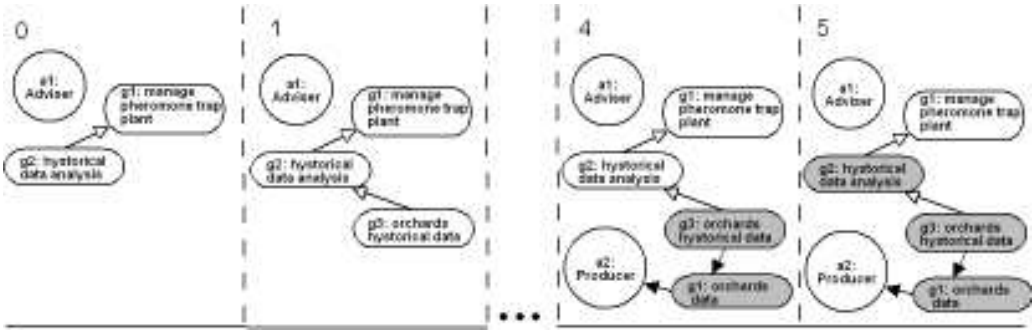

Fig. 6. An example of a frame sequence. Frame 2 to 3 have been omitted. A gray goal is a fulfilled goal. Going from frame 4 to frame 5 we can notice how goal fulfillment propagates from the goal in the dependency towards the original goal, along the goal decomposition chain. The example is taken from [9].

syntax of the modeling language has been defined through a metamodel, described in [2]. The diagram depicted in Fig. 5 gives a graphical representation of goal analysis in a model which includes two actors: the actor "Advisor" and the actor "Producer" representing two stakeholders of the application domain of interest [1]. The actor "Advisor" depends on the actor "Producer" for the achievement of the goal to acquire "orchards data". The dashed balloon includes the analysis of the goal "manage pheromone trap plant" conducted from the point of view of the actor "Advisor".

The *Tropos* diagram also contains implicit information about creation and fulfillment relations between a model's entities. For example, a subgoal can be created only after the creation of an upper level goal, and a decomposed goal can be fulfilled only if all the subgoals (*and* decomposition) or at least one subgoal (*or* decomposition) is fulfilled.

The *Tropos* model can be attached with formal specifications in temporal logic. The specifications can be checked using model checking techniques [10].

The diagram in Fig. 6 depicts a *frame sequence* which expresses a desired behavior of the system modelled in Fig. 5. Specifying the frame sequence, an analyst would like to validate the previously specified model with respect to a specific scenario. That is to ensure that the goal "historical data analysis" of the actor instance "a1" (of type "Advisor") can be fulfilled by means of the actor instance "a2" (of type "Producer") who commits itself to the satisfaction of the "a1"'s goal of acquiring "orchards data".

[1] The domain is that of Integrated Production in Agriculture. The example is taken from [9].

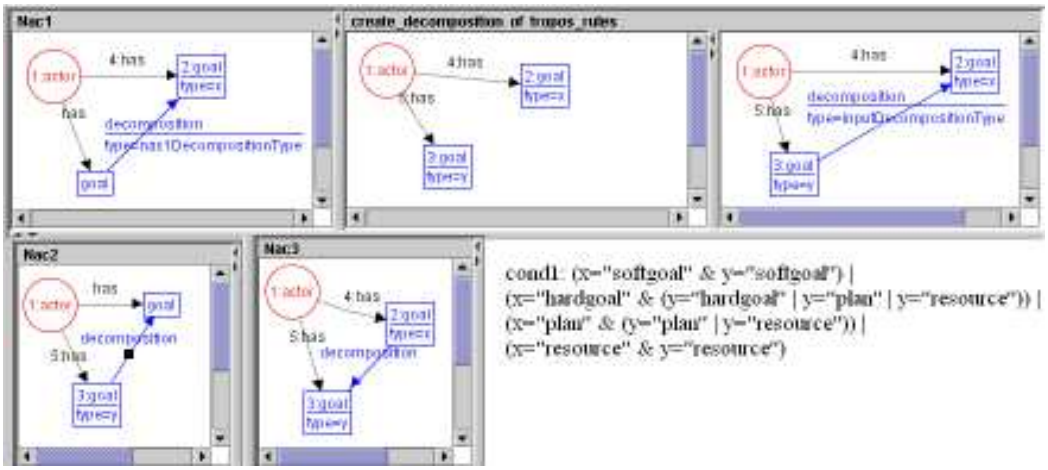

Fig. 7. An example of rule implemented in AGG. The rule specifies the creation of a decomposition link.

Further information on *Tropos* and on its application to different case-studies can be found in [11].

3 Visual modeling in *Tropos* and Model Driven Architecture

3.1 The objectives

Model Driven Architecture approach concentrates on later phases of software development, namely, on architectural and detailed design of a software system. However earlier phases like requirements are also very important because the phases are most error prone, and correction of the errors in the later phases is very expensive. Therefore we think that it is very important to support MDA for the earlier phases of software development. *Tropos* methodology concentrates on requirements modeling starting from analysis of the system-to-be [2]. We interpret MDA approach as visual modeling activity where more abstract models are refined to more detailed using transformation techniques. Thus the approach can be applied to different paradigms (i.e. Object Oriented or Agent Oriented), different methodologies (i.e. Rational Unified Process or Tropos) and different phases (i.e. requirements, architecture, design). Automatic tools should support the consistency between different types of models. In this vision, the following activities should be supported with automated tools for *Tropos* methodology:

- different views of the *Tropos* model should be conforming between each other: *Tropos* diagram, sequence diagram and formal specifications;
- models of different phases of the modeling process should be consistent between each other, there should be requirements traceability and automatic propagation of changes from model of one phase to models of other phases;

In the rest of this paper we are going to review how our Graph Rewriting based approach of visual modeling for *Tropos* can satisfy the requirements and assure compliancy of *Tropos* methodology with MDA.

3.2 Graph rewriting rules for Tropos modeling

A *Tropos* model can be represented as a graph whose nodes represent entities like actors, goals, resources, plans, and whose edges represent the relationships that can be defined among these entities, such as dependencies and decomposition relationships. Both nodes and edges have attributes which are used to complete the entity specification. The attributes can be also the parts of formal specification which can be extracted later from the graph model. It seems very reasonable to represent the syntax of *Tropos* using a set of rules that allow for generating all valid *Tropos* models.

We implemented these GR rules with AGG. The syntax requirements are expressed as positive application conditions (left side of the rules), NACs and conditions on attributes. The graph representation of a *Tropos* model used by AGG makes explicit some relationships

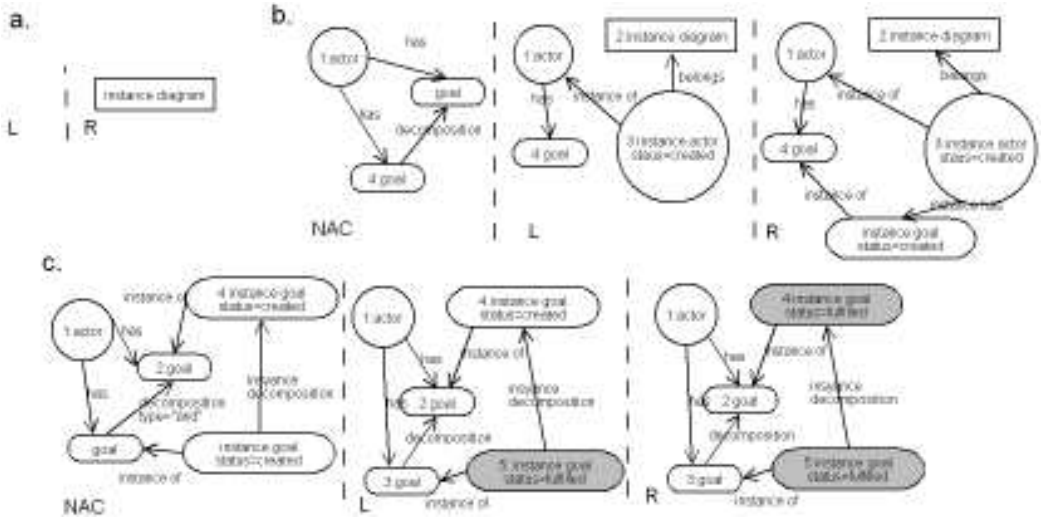

Fig. 8. The example of the rules for instance creation: a) the rule creating an instance diagram label; b) the rule for creating a subgoal; c) the rule marking fulfillment of decomposed goal.

between entities that are implicit in *Tropos* diagrams, such as the one depicted in Fig. 5, by means of additional edges/nodes or attributes. For instance, the fact that a goal belongs to an actor (the goal is inside of a dashed balloon attached to the actor in Fig. 5) is expressed with an edge *has* from the actor node to the goal node. Different types of intentional entity (*hardgoal, softgoal, plan, resource*) are represented as attributes of node of type *goal*. This allows to reduce considerably the number of the rewriting rules. All the additional information, like names and parts of formal specification, associated to a node are represented as attributes as well.

Fig. 7 contains an example from the rule-set for building a *Tropos* model. The example is going to be used for the explanation of how we ensure some syntax requirements using GR rules. The input for the rule is the type of the decomposition link (*or/and*). The left side of the rule (positive application condition) guaranties the application of the rule only to two goals belonging to the same actor. The NACs in the rule ensure that we will have the same type of decomposition link from the same root goal and that a decomposition link between the two goals has not been already created. Moreover, it forbids to have a goal that is a child of two different goal decompositions. The condition on attributes "cond1" controls that the source and the target in a decomposition are compatible, according to the *Tropos* language syntax given in [2].

Among the advantages of adopting this approach we'd like to mention the following:

- Flexibility of the modeling system. Let's suppose we change some syntax requirement or introduce new language constructs. In this case we have to change or add NACs to the rules or add new rules, which is a limited effort.
- Guiding the modeling process. We can exploit the functionalities provided by AGG such as suggesting possible rule mapping completions, for instance to highlight possible completion objects. In the case of the rule for creating a decomposition link, the analyst can select the goal to be decomposed and the tool can point out all the possible subgoals that are valid completions for the rule.

3.3 Building Tropos instances and frame sequences

A *Tropos* model represents domain actors, goals and dependencies and it may include, as well, instances of those entities. The process of building instances requires an additional set of rules that can be applied to the same model. The set of rules for creating instances has to take the fulfillment/creation dependency information (described in Sec. 2.3) into account as rules conditions.

Fig. 8 depicts an example of rules of this kind. There are two new types of links introduced in the rule: *instance of* link means that the object is an instance of a *Tropos* diagram object, the link *belongs* means that an actor instance (and thus all its goal instances) belong to an

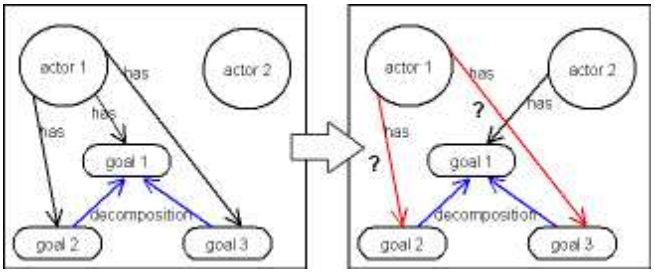

Fig. 9. The movement of a goal to another actor.

instance diagram.

The rule Fig. 8a creates a new node of type *instance diagram* and thus a new instance diagram. The rule in Fig. 8b creates an instance of goal. The NAC for this rule ensures that the goal is not a subgoal. (The rule for a subgoal creation has not been described in the Fig. 8.) The rule in Fig. 8c changes the status of an upper level goal to "fulfilled" only if all the subgoals are fulfilled (*and* decomposition) or at least one subgoal is fulfilled (*or* decomposition).

We remark that an instance diagram is a snapshot of a frame sequence (see Fig. 6). We can reproduce a frame sequence for it by reapplying the set of rules for the instance building. Every frame corresponds to a step of the instance building process which is performed by the analyst starting from the rule *instance diagram* as a first, empty frame.

The main advantage of the approach here is that it guaranties to build instances within a *Tropos* model. Using the set of rules for the instances we can create any number of instances for the *Tropos* diagram. And, so that all of them are guarantied to be compatible with the model itself.

3.4 Derivation transformations

The GR rules described above include only rules for creating new elements. While additional rules can be added for some deletion and changing activities it looks like several of these activities can not be done with only one graph rewriting rule. There are two reasons why: first a deletion step may require many preconditions (e.g., we can delete an actor only after having deleted all the goals belonging to it); second, a rewriting step may impact to a large subset of the model (e.g., movement of a goal from an actor to another involves movement of all the dependent subgoals, Fig. 9). For such kind of diagram modifications we foresee a derivation transformation approach as the one described in [12,13]. This will provide a general, semantically found solution to the problem.

For the moment, we are experimenting with a more naive approach which is based on the idea of keeping and saving the derivation history and of applying external, hand-written algorithms to represent specific derivation transformations. For example such an algorithm should support goal movement from one actor to the other. It should change the rule for creation of the goal replacing the original actor with the desired one, it should check all the dependency links and all the subgoals, propagating then the same rule transformation to the subgoals. All the other links, except decomposition, should be destroyed. The new derivation can be reapplied resulting into the changed diagram. Analogous updating will be done for all the instance diagrams. The same technique can be used for other type of restructuring in a model.

Some of the derivation transformation algorithms can require the analyst to input his/her decision. For example a decision on what the analyst is going to do with a specific type of link in the diagram while moving a goal from one actor to another, or on how the analyst would like to rename a goal.

3.5 Different phases consistency

Visual models obtained during different phases of the development process like early and late requirements, architectural and detailed design, should be consistent between each other. Early and late requirements and architectural design phases of *Tropos* methodology use and refine the same visual model, specifying it, and adding new details. We are going to use the

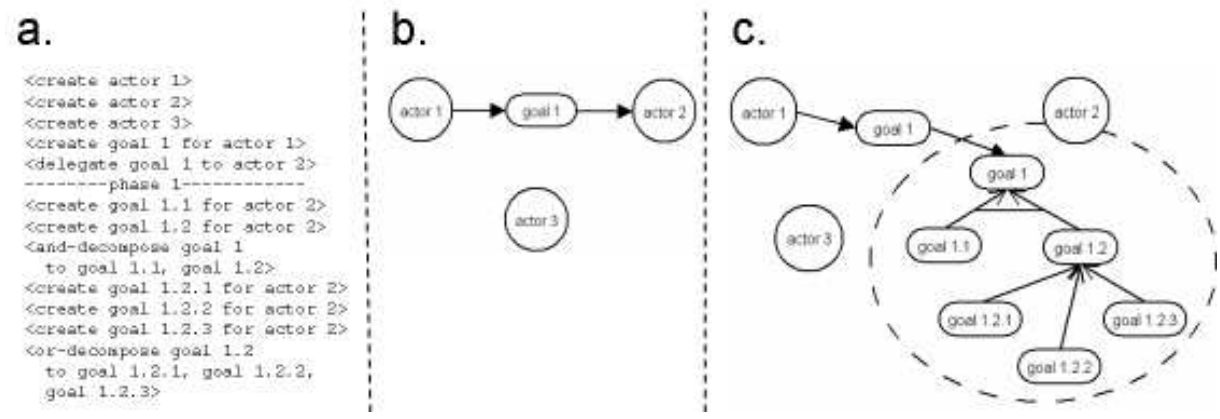

Fig. 10. The example of movement of a goal from one actor to another, **before** the "move goal" reconfiguration: a. - the corresponding derivation history with the label for the phase 1; b. - the model for the phase 1; c. - the model for the later phase.

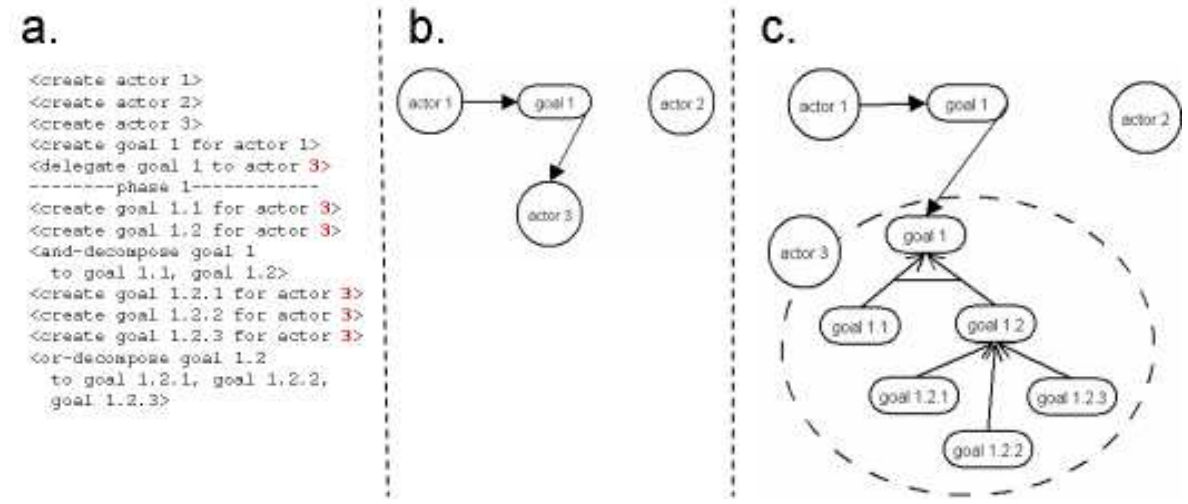

Fig. 11. The example of movement of a goal from one actor to another, **after** the "move goal" reconfiguration: a. - the corresponding derivation history with the label the for phase 1 (the changes are marked in bold-red font); b. - the model for the phase 1; c. - the model for the later phase.

same derivation history for all the three phases of the development process. In this way the whole derivation history can contain information of all the models of different phases. Thus, the derivation transformation approach for reconfigurations can ensure the consistency of the models. An example of such kind of reconfiguration can be the example of movement of the *goal 1* from *actor 2* to *actor 3* depicted in Fig. 10 and Fig. 11. In the general derivation history (e.g. Fig. 11.a) you can see the label *phase 1* which means that the part of the derivation history above the label corresponds to the model of *phase 1*. The derivation transformation algorithm replaces *actor 2* to *actor 3* at all the rules for creating goals that belong to the decomposition tree of *goal 1*. In the Fig. 11 both models for *phase 1* and the *later phase* are reconfigured because they are constructed within the same derivation history. Therefore the changes made in one phase will propagate to another phases. This fact means that the models of all the phases are consistently modified during a change of one of them.

4 Conclusion and future work

In this paper we have analyzed how our GR based approach of *Tropos* visual modeling coincides with MDA. Graph Rewriting techniques together with derivation transformations can assure:

- conformance of different views of the *Tropos* model: *Tropos* diagram, sequence diagram and formal specifications;
- consistency of models on different phases of the modeling process, providing requirements traceability and automatic propagation of changes from model of one phase to models of other phases;

An automatic tool based on the ideas described in the paper can support MDA approach for Tropos methodology, and MDA can be extended to early phases of software development process like requirements analysis.

We are currently defining an appropriate notation for representing derivation history and completing the implementation and validation of the described GR techniques in AGG.

The work we have presented in the paper covers only the requirements phase. We intend to push forward the usage of graph transformations also in the later phases (in particular architectural design and detailed design) following the guidelines of the *Tropos* methodology [2].

A long term objective is that of integrating GR into a CASE tool aimed at supporting visual modeling within the *Tropos* methodology. The GR based CASE tool is going to be compliant with the MDA ideology.

References

[1] A. Brown. An introduction to Model Driven Architecture Part I: MDA and todays systems. *IBM*, 2004.

[2] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 2003. In Press.

[3] M. Pistore A. Novikau, A. Perini. Graph Rewriting for Agent Oriented Visual Modeling. *GT-VMT'04 workshop, Barcelona, Spain, 27-28 aprile*, 2004.

[4] OMG group. *Unified Modeling Language specification*. Current version: 1.5. 2003. See also UML OMG site: http://www.omg.org/uml/.

[5] E. D. Willink. UMLX: A graphical transformation language for MDA. *GMT Consortium*, 2003.

[6] A. Agrawal G. Karsai. Graph Transformations in OMGs Model-Driven Architecture. 2003.

[7] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic Approaches to Graph Transformation. Part 1: Basic Concepts and Double Pushout Approach. *Handbook of Graph Grammars and Computing by Graph Transformation*, 1: Foundations, 1997.

[8] C. Ermel, M. Rudolf, and G. Taentzer. The AGG approach: Language and environment. In *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2: Application, Languages and Tools. World Scientific, 1999. See also AGG site: http://tfs.cs.tu-berlin.de/agg/.

[9] A. Perini and A. Susi. Developing a Decision Support System for Integrated Production in Agriculture. *Environmental Modelling and Software Journal*, 2003. Submitted to.

[10] A. Fuxman, M. Pistore, J. Mylopoulos, and P. Traverso. Model Checking Early Requirements Specifications in Tropos, 2001.

[11] Tropos Project site: http://www.troposproject.org/.

[12] D. Hirsch and U. Montanari. Two Graph-Based Techniques for Software Architecture Reconfiguration. In M. Bauderon and A. Corradini, editors, *Electronic Notes in Theoretical Computer Science*, volume 51. Elsevier, 2002.

[13] D. Hirsch and U. Montanari. Higher-Order Hyperedge Replacement Systems and their Transformations: Specifying Software Architecture Reconfigurations. In H. Ehrig and G. Taentzer, editors, *Proceedings of the Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems (GRATRA 2000)*, pages 215–223, 2000.