Available online at www.sciencedirect.com
SCIENCE DIRECT
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 109 (2004) 85–96
www.elsevier.com/locate/entcs

Graph Rewriting for Agent Oriented Visual Modeling

Aliaksei Novikau, Anna Perini [1]

ITC-Irst, Via Sommarive, 18, I-38050 Trento, Italy

Marco Pistore[2]

University of Trento, Via Sommarive 14, I-38050 Trento, Italy
ITC-Irst, Via Sommarive, 18, I-38050 Trento, Italy

Abstract

In this paper we will describe our approach, based on Graph Rewriting (GR), to support the visual modeling process in *Tropos*, an Agent Oriented software engineering methodology. We will give examples of the rewriting rules which specify the syntax of *Tropos* and will discuss how this graph rewriting rule-set can support an analyst in building correct models (that is models consistent with the *Tropos* language). Moreover we will consider how GR permits to adopt modeling language variants in a flexible way and to support an incremental modeling process.

Keywords: visual modeling, graph rewriting, agent-oriented modeling, AGG.

1 Introduction

Visual modeling is a common practice in software development, especially in Object Oriented software development where a standard modeling language (the Unified Modeling Language - UML [1]) has been proposed and several CASE tools at support of the modeling activity are available [2]. This led also to the adoption of visual modeling as a core discipline in industrial software development processes, such as the Rational Unified Process [3]. Nevertheless,

[1] Emails: novikau@irst.itc.it, perini@irst.itc.it
[2] Email: pistore@dit.unitn.it

doi:10.1016/j.entcs.2004.02.058

some interesting research problems are still open, such as how to provide a common semantics for structural and behavioral UML diagrams [4,5], how to enhance OCL constraints for UML class diagrams with Graph Rewriting (GR) concepts [6]. GR plays an important role in solving some of them, as discussed in previous GT-VMT workshops [7].

Agent Oriented (AO) software engineering is a paradigm that has been recently proposed (see for instance [8] and previous editions) with two main meanings, namely, *(i)* as an approach to engineering Multi-Agent Systems and *(ii)* as an approach to developing distributed systems based on the agent paradigm for the analysis and the design of such systems [9]. Visual modeling plays a crucial role also in AO approaches. We refer, in particular, to the *Tropos* methodology [10], an agent oriented software engineering methodology (according to definition *(ii)*), that is based on a visual modeling language and provides a set of analysis techniques. In particular, specific *Tropos* diagrams allow to represent different aspects of a model structure, while dynamic aspects of the model can be specified through formal annotations expressed as temporal logic formulas [11].

We are focusing on visual modeling in *Tropos* and on a set of specific issues related to the development of a CASE tool at support of visual modeling in *Tropos*, such as, how to assure that a model is correct with respect to a given syntax, how to support the building of consistent instances of a given model, and how to provide services for automatic restructuring of the model like the application of different patterns and refactoring.

In this paper we discuss how GR could be used to define a solution to these problems.

The paper is structured as follows: in Sec. 2 we will give a brief description of the *Tropos* modeling language and present the set of problems related to the development of a CASE tool for visual modeling that we intend to address exploiting GR; we will also recall the basic features of GR that we will use in the following. In Sec. 3, we will describe our approach, namely, the set of graph rewriting rules for building a *Tropos* model; a set of rules for supporting instance creation and preliminary ideas on how to manage the consistent changing or restructuring of a model. Related works are briefly discussed in Sec. 4. Finally, conclusions and future work are presented in Sec. 5.

2 Background

2.1 Tropos methodology

Tropos [10] is an AO methodology for building distributed software systems which provides a visual modeling language. Visual modeling is used for the

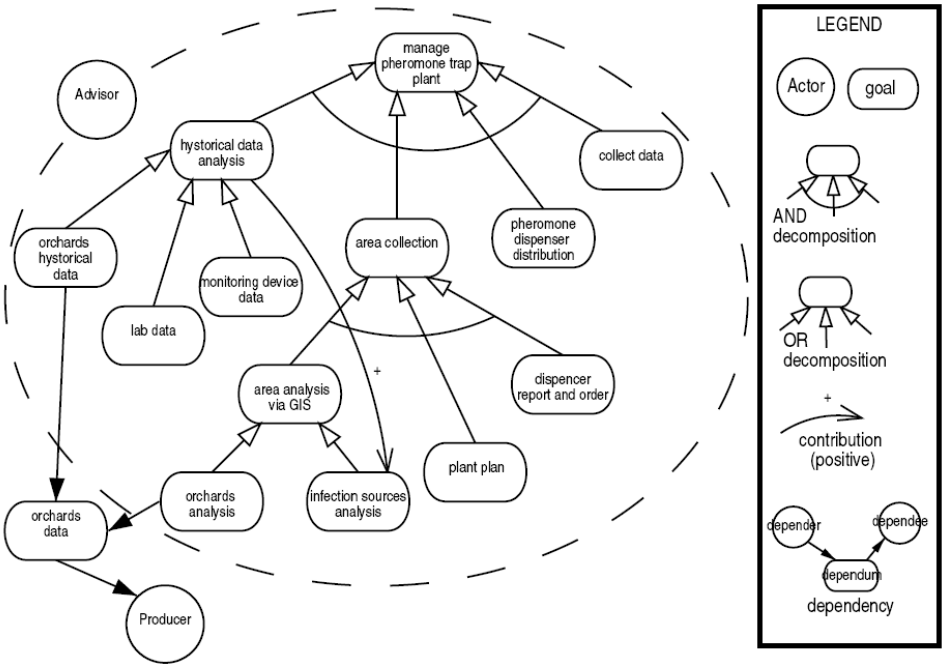

Fig. 1. An example of *Tropos* actor and goal diagram.

analysis of the application domain as well as for the requirements, the architectural, and the detailed design of the system-to-be. Among the basic concepts provided by the language: the concept of *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 *goal* for representing the strategic interests of an actor; the concept of *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 syntax of the modeling language has been defined through a metamodel, described in [10]. *Tropos* offers a set of analysis techniques, such as *and/or decomposition* of goals, *means-end* analysis and *contribution* analysis and a set of diagrams for visualizing model properties. The diagram depicted in Fig. 1 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 [3]. 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

[3] The domain is that of Integrated Production in Agriculture. The example is taken from [12].

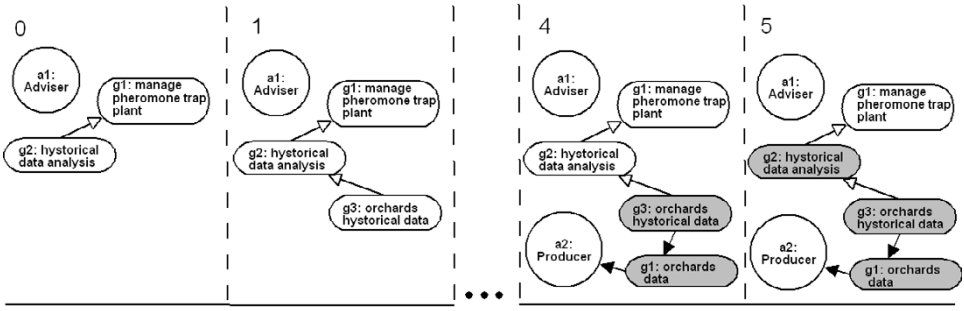

Fig. 2. An example of frame sequence. Frame 2 to 3 have been omitted. A gray goal is a fulfilled goal. Going from frame 4 to frame 5 we can notice how goal fulfillment propagates from the goal in the dependency towards the original goal, along the goal decomposition chain.

plant" conducted from the point of view of the actor "Advisor". Examples of *and/or decomposition* as well as *contribution* analysis are shown.

The *Tropos* diagram contains also implicit information about creation and fulfillment relations between 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 diagram in Fig. 2 depicts a frame sequence which expresses a desired behavior of the system modeled in Fig. 1. 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 herself for the satisfaction of the "a1"'s goal of acquiring "orchards data".

Further information on *Tropos* and on its application to different case-studies can be found in [13].

2.2 Supporting Tropos visual modeling

Here we focus on a set of specific issues related to the development of a CASE tool which supports the analyst in the process of building and revising a visual model in *Tropos*.

The output of a visual modeling process should be a model which is correct with respect to the syntax of the modeling language. However this process goes through the specification of partially correct models. An example could be a creation of a goal without an actor owning it (while according to the language every goal belongs to an actor). Such model should be considered as non completed and the analyst should be warned about this. Moreover, the

analyst should be guided towards possible model completions. For example when the analyst starts decomposing a certain goal, the tool should highlight all the goals that can be part of the decomposition. The analyst should also be informed why a building step is forbidden. For instance, if the analyst attempts to create an actor with an already existing name, a clear message should point out the error.

Building a model instance can be helpful for the analyst, for instance, in order to check if the model includes a specific scenario (or behavior). Supporting the analyst in building a consistent instance of a model means that in case she wants to mark "orchards historical data" in frame 1 of Fig. 2 as fulfilled (colored goals) then the tool should warn that it is impossible because the goal should be delegated to the "Producer" actor which should fulfill it then (e.g. the correct instance is the one depicted in frame 5 of Fig. 2).

The tool should be easily adaptable with respect to language variants, such as using only a subset of the modeling language or an extended language where new elements or syntax requirements have been added (for example a new type of decomposition links in addition to *and/or*).

The tool should support also automatic restructuring of the model such as the application of different patterns and refactoring. For instance, if the analyst changes the actor and goal diagram depicted in Fig. 1, then the associated instance diagrams should be consistently updated. An example can be a change of a "historical data analysis" decomposition type from *or* to *and*, which means that in order to fulfil the goal all the subgoals should be fulfilled. So the frame 5 would be no more valid and would require the creation and fulfillment of another subgoal. Another example of model restructuring can be the movement of a goal and all its subgoals from one actor to another. More generally, restructuring process should guarantee consistency between different views of resulting model.

2.3 GR techniques and tools

For solving the problems described above we will use GR. In particular, we consider promising to exploit attributed GR which allow to keep some relevant diagram information in terms of a set of attributes attached to nodes and edges of a graph representing the model. Moreover, the rewriting system should support complex conditions on rules application such as Negative Application Conditions (NACs) and conditions on attributes.

Preliminary evaluation of our ideas have been conducted using a system which implements GR techniques called AGG (Attributed Graph Grammar System) [14]. AGG is a rule based visual language written in java supporting an algebraic approach to graph rewriting. It provides flexible attributed

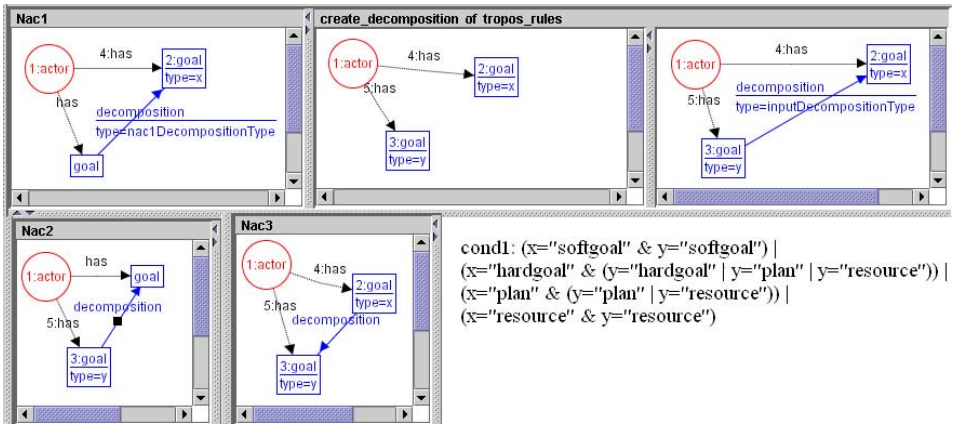

Fig. 3. An example of rule implemented in AGG. The rule specifies the creation of a decomposition link.

graphs and generic application conditions including NACs and attribute conditions.

3 The approach

We describe our approach along three basic steps: in Sec. 3.1 we describe a first step in applying GR for guiding the analyst in designing valid models; in Sec. 3.2 we address the problem of supporting instance building; in Sec. 3.3 we point out techniques needed to provide restructuring/updating capabilities.

3.1 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. It is hence 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 between entities that are implicit in *Tropos* diagrams, such as the one depicted in Fig. 1, 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. 1)

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. 3 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 [10].

Among the advantages of adopting this approach to address the questions posed in Sec. 2.2 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.
- Build partially correct models. In order to allow for partially correct models (e.g. models with goals not associated to actors) we can extend the set of types for building a *Tropos* models to include terminal and nonterminal types. The model is considered to be partially correct if the graph contains nonterminal typed elements. Only graphs containing terminal elements represent correct models. An example of rule involving nonterminal elements is shown in the Fig. 4. The rule "a." creates a goal without actor. So, we are using a nonterminal type for such kind of goal: $< goal >$ which is an incorrect entity according to the *Tropos* syntax because it doesn't belong to an actor. A $< goal >$ can not be further analyzed, e.g. it cannot be decomposed. The second rule "b." allows to assign an actor to the nonterminal

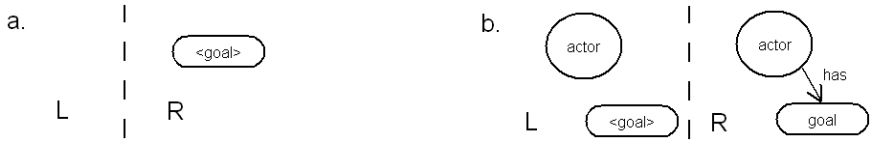

Fig. 4. The example of rules for managing nonterminal elements.

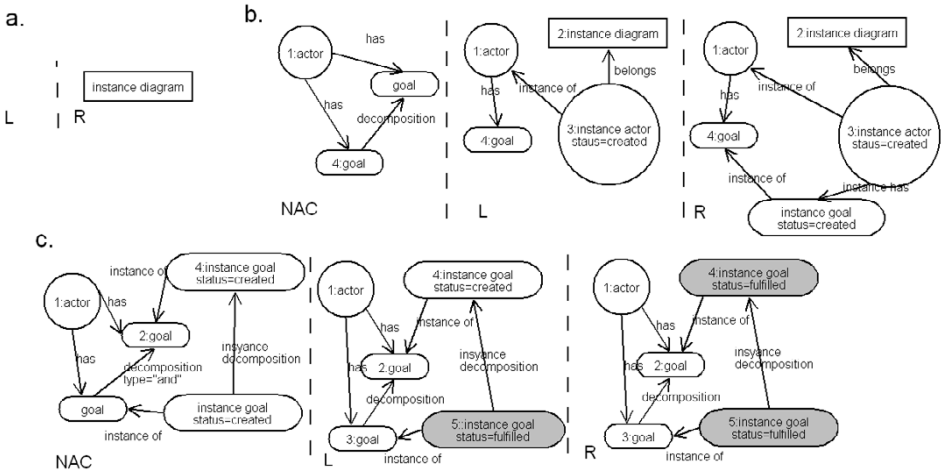

Fig. 5. The example of the rules for instance creation: a) the rule creating an instance diagram label; b) the rule for creating a subgoal; c) the rule marking fulfillment of decomposed goal.

goal. Here the nonterminal element $< goal >$ is replaced with the terminal element *goal*.

3.2 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. 2.1) into account as rules conditions.

Fig. 5 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 instance diagram.

The rule Fig. 5a creates a new node of type *instance diagram* and thus a new instance diagram. The rule in Fig. 5b 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. 5.) The rule in Fig. 5c changes the

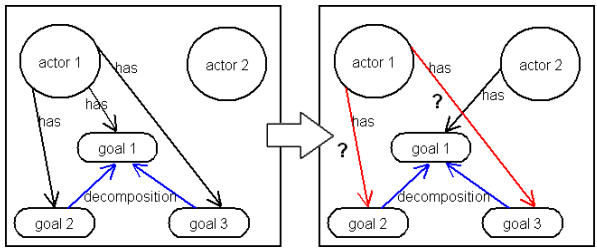

Fig. 6. The movement of a goal to another actor.

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. 2). 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.

3.3 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. 6). For such kind of diagram modifications we foresee a derivation transformation approach as the one described in [15,16]. 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.

4 Related work

In the field of GR for visual modeling there are different directions, and among them OO modeling with UML is gaining a lot of attention. In particular, several works address issues of consistency and semantics of the different types of UML diagrams. Some works on consistency of UML class and sequence diagrams use attributed GR for checking of consistency of the two types of diagrams [4]. In [5] an approach is described for an integrated graph based semantics for UML diagrams such as class, object and state diagrams. The idea of the paper is that the analyst associates graph rewriting rule to the operations of a class in a class diagram. Using these hand-written rules and other rules generated from statechart diagrams, it is possible to check consistency of object, sequence, and collaboration diagrams. Also in our approach we are using GR as a basis for consistency between static and dynamic views of a *Tropos* model. However, our approach does not require the analyst to write rules by hand, since a predefined set of rules is sufficient to support instantiation taking into account *Tropos* language semantics.

Among the available tools that implement GR and that offer also advanced editing functionalities we considered GenGEd or DiaGen both described in [17]. We preferred to use the AGG tool since our focus is on providing a graph transformation engine. This engine shall then be integrated in a CASE tool which provides a broader set of functionalities, and that is being developed in our research group.

5 Conclusion and future work

In this paper we have described our approach, based on GR, to support the visual modeling process in *Tropos*. We have given examples of the *Tropos* rewriting rules which specify the syntax of the *Tropos* language and supports an analyst in building correct models. These rules have been implemented with AGG. The GR approach gives flexibility in adopting modeling language vari-

ants and can be adapted to allow for the definition of partially correct models. Moreover, derivation transformation seems to be a promising way to support the restructuring of a model in consistent ways. We are currently defining an appropriate notation for representing derivation history and completing the implementation and validation of the described 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 [10]. In this extension we will refer also to the work done in [18] on a formal agent-oriented modeling methodology based on UML and GR that covers different activities ranging from requirements specification to analysis and design.

A long term objective is that of integrating GR into a CASE tool aimed at supporting visual modeling within the *Tropos* methodology.

References

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

[2] CASE tools list odered by names: http://www.cs.queensu.ca/Software-Engineering/tools.html.

[3] Rational Unified Process product overview: http://www-306.ibm.com/software/awdtools/rup/.

[4] A. Tsiolakis and H. Ehrig. Consistency Analysis of UML Class and Sequence Diagrams using Attributed Graph Grammars. In H. Ehrig and G. Taentzer, editors, *Proc. of Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems, Berlin, March*, 2000.

[5] M. Gogolla, P. Ziemann, and S. Kushke. Towards an Integrated Graph Based Semantics for UML. In *Electronic Notes in Theoretical Computer Science*, volume 72. Elsevier, 2003.

[6] A. Schürr. Adding Graph Transformation Concepts to UML's Constraint Language OCL. In *Electronic Notes in Theoretical Computer Science*, volume 44. Elsevier, 2001.

[7] GT-VMT02 workshop site: http://www2.cs.fau.de/GTVMT02/.

[8] F. Giunchiglia, J. Odell, and G. Weiss, editors. *Agent-Oriented Software Engineering III*, volume 2585 of *LNCS*. Springer-Verlag, 2002. Third International Workshop, AOSE 2002, Bologna, Italy, July 15, 2002.

[9] E. Yu. Agent-Oriented Modelling: Software versus the World. In *Agent Oriented Software Engineering*, pages 206–225, 2001.

[10] 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.

[11] A. Fuxman, L. Liu, M. Pistore, M. Roveri, and J. Mylopolous. Specifying and Analyzing Early Requirements: Some Experimental Results. *RE-2003, the 11th IEEE International Requirements Engineering Conference*, 2003.

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

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

[14] 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/.

[15] 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.

[16] 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.

[17] R. Bardohl, G. Taentzer, M. Minas, and A. Schürr. Application of Graph Transformation to Visual Languages. In *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2: Application, Languages and Tools. World Scientific, 1999.

[18] R. Depke, R. Heckel, and J.M. Küster. Formal Agent-Oriented Modeling with UML and Graph Transformation. In *Sci. Comput. Program.*, volume 44, pages 229–252. Elsevier, 2002.