A Design Framework for Generating BDI-Agents from Goal Models

Loris Penserini, Anna Perini, Angelo Susi, Mirko Morandini, John Mylopoulos
ITC-IRST
Via Sommarive 18,
I-38050, Trento - Povo, Italy
{penserini,perini,susi,morandini}@itc.it, jm@cs.toronto.edu

ABSTRACT

We define a tool-supported design framework that allows to specify an agent goal model and to automatically generate fragments of a BDI agent from it. We devise the design process as a transformation process from platform-independent design models to platform-specific models and then to code. The design framework is demonstrated by referring to the *Tropos* methodology and to the JADE/Jadex platform. In this short paper, key steps in the process are illustrated through an example.

General Terms

Design

Keywords

Agent-Oriented Software Engineering

1. INTRODUCTION

Goal models have been used in distributed Artificial Intelligence as a means for capturing agent intentions and guiding agent coordination [4, 6, 7]. These goal models consist of goal graphs whose nodes represent goals. Goals can be related through AND/OR relationships that represent the hierarchical decomposition of a goal into simpler goals. In addition, goals can be related through different kinds of inter-dependency links that represent conflicts between goals, or resources needed for the fulfillment of inter-dependent goals. In this context, goal models guide software agent choices (behavior) at run time. Similar goal models, (*GM* from now on) have been adopted, by so called *goal-oriented* approaches to software (requirements) engineering [1, 3, 10]. In this context, a *GM* allows a designer to represent and reason about stakeholder goals in a given application domain in order to derive requirements for a system-to-be. According to these approaches, *GM*s provide analysis and design artifacts during system development. *GM*s can also give support in exploring and evaluating alternative solutions which can meet stakeholders expectations (goals) and in detecting conflicts that may arise from multiple viewpoints. Some approaches adopt a formal notation which enables model-checking verification of the resulting models [3, 1].

Taking advantage of the above results, we propose to use *GM*s at different abstraction levels in engineering Multi-Agent Systems (MAS), namely at design- and at run-time. A *GM* at design time represents the purposes behind a MAS, making the dependencies between system agent goals and stakeholder goals explicit. Knowledge level concepts such as those of *agent*, who can be social, organizational, human or software, *goals*, and *social dependencies* for defining the obligations of agents to other agents are used at this level. Moreover, a view on the system behavior can be obtained by querying a design-time *GM*. Let's consider a MAS supporting different word searching techniques over the Internet. We may query our design-time *GM* to determine the alternative ways the system can manage a request (event) by a user. For instance, finding grammatical/semantic information about a word might be accomplished by either using a search engine such as google, or by looking up an on-line dictionary.

The main objective of this paper is to propose a tool-supported design process that takes as input such *GM*s and generates fragments of a BDI agent. These fragments include goals and capabilities, along with a reasoning strategy for selecting and running appropriate capabilities, given a goal and a set of domain conditions. Our approach offers a systematic process for operationalizing a *GM* into a set of capabilities and for automating BDI-agent code generation from the *GM* design artifacts.

This approach aims at addressing crucial issues in developing and maintaining complex distributed software. Moreover, we believe it offers an interesting direction towards engineering adaptive systems, given that *GM*s can be extended (modified) at run-time [7] and that we can provide traceability links between *GM*s at different levels of abstractions. An issue that seems still understudied by the main Agent-Oriented Software Engineering (AOSE) methodologies [5, 2]. In this short paper, we sketch our tool-supported design framework, by first giving an example of design artifact, developed using the *Tropos* AOSE methodology [1], along with key modelling concepts, such as the concept of *GM* and agent capability (Section 2). We then focus on code generation (Section 3). The JADE/Jadex MAS platform is considered for implementation.

2. GOAL MODEL DESIGN

The *Tropos* agent-oriented methodology [1] borrows modelling and analysis techniques from goal-oriented requirements engineering frameworks and integrates them into an agent-oriented paradigm. A core activity along this process is conceptual modelling. The modelling language offers concepts of *actor*, *goal*, *plan*, *resource*, *capability*, and of *social dependency* between actors for goal achievement, a graphical notation to depict views of a model, analysis techniques and supporting tools[1]. Adopting *Tropos* in our

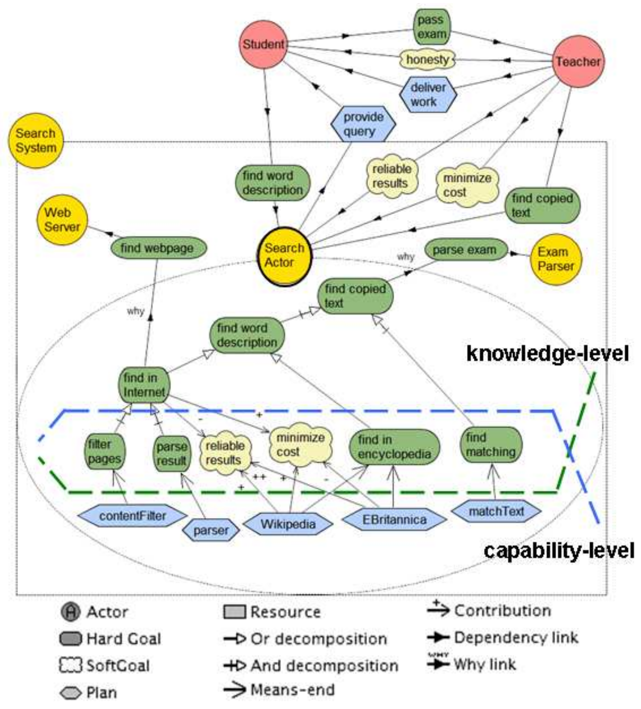

Figure 1: *Tropos* architectural design: Agent knowledge and capability levels.

framework allows us to represent and reason on a *GM* resulting from the analysis of each actor's point of view. Specifically, a *GM* in *Tropos* is represented in terms of a forest of AND/OR goal trees, along with lateral contributions labelled $+$,$++$ (i.e. if $g_1 \xrightarrow{++} g_2$ and g_1 is fulfilled, so is g_2) and $-$,$--$ (if g_1 is fulfilled, g_2 is denied) and means-ends relationships among goals and plans. Examples are depicted in Fig. 1, which illustrates a fragment of a goal-oriented *Tropos* specification of the search system's requirements. The system is intended to support students and teachers in exam related activities. Let us consider the goal dependencies find the word description and find copied text, between the actors Student and Teacher and the agent role Search Actor, which model the possibility for such goals to be triggered by a request message from the users (or a personal agent of) Student and Teacher. In other words, to pass an exam a student has to deliver some written homework, while the teacher wants to evaluate originality of the student homework, for instance, checking if the student copied from existing material (e.g. encyclopedia, or Internet).

In particular, we will point out the two different abstraction levels that characterize the agent design, *knowledge level* and *capability level*. The knowledge level gives to the agent a picture of the real world in terms of goal concepts that describe alternative ways to cope with complex and simple problems. For example, goals may be naturally triggered by agent internal or external events, e.g. a FIPA-request message (i.e. an external event) can carry out information about what goals are required to be achieved by the receiver agent. The capability level is composed of leaf node goals that are satisfied by root-level plans through the *Tropos* means-end relationship [9].

[1]For more details on modelling activities and supporting tools see [1, 9].

Although this paper focuses on the knowledge level, for the concept of *capability* we adopt the revised definition proposed in [9], which distinguishes the concept of *ability* from the concept of *opportunity* and introduces a specific notation to model them. The *ability* component refers to plans for achieving a given goal and is specified in *Tropos* by a means-end relationship between the goal and the plan, while the *opportunity* component represents user preferences and environmental conditions, which may, at run-time, enable or disable the execution of the ability component.

Capability modelling starts during requirements analysis by identifying agent capabilities and their correlations with stakeholder needs. Tab 1, depicts some of the capabilities that operationalize the behaviour of the *GM* of the Search Actor in the example given in Fig. 1.

Capab.	**Means_End(goal,plan)**	**List of Contributions**
cp_1	`filter pages, contentFilter`	`{}`
cp_2	`parse result, parser`	`{}`
cp_3	`find in encyclopedia, EBritannica`	`{reliable result ++}` `{minimize cost -}`
…	...	...

Table 1: Search Actor **capabilities.**

3. TOWARDS AUTOMATED CODING

In this section we sketch the transformations which are preliminary to the automated coding of *GM*s into BDI agents. We direct the interested reader to [8] for a complete description of the process and some experimental evaluation.

Agent code generation requires a transformation of the *Tropos GM* specification into a specific one for the implementation platform, i.e. Jade[2] agents in our case.

The ability parts of *Tropos* capabilities, are mapped to UML activity-diagrams, through model-driven automatic transformation techniques, and are enriched with sequence diagrams to specify execution workflow and interaction protocols respectively (detailed can be found in [9]). The rest of the *GM*, including the capability's opportunity part, relies on an automatic transformation from *Tropos* models to Jadex BDI agents, as detailed in the following.

We first describe the semantics of the adopted sub-set of *Tropos* concepts, and then give an overview of some of the proposed mappings between such concepts and related data-structures of a Jadex BDI agent. The proposed approach considers *Tropos* agent *GM*s at architectural design. The specification for the mapping process has been conducted along two phases: basic concept mappings (goals, softgoals, plans, resources) and structure mappings (AND/OR goal dependencies, means-end links, contribution links, delegation and dependency links) as detailed below.

Goal. As a Jadex-goal can only be triggered by a Jadex-plan, hence a *Tropos* goal is directly mapped to a pair of $< goal, plan >$ in Jadex.

Softgoal. In our prototype, they are mainly used to define opportunities for the selection of the next goals or plans to pursue along the *GM*. A softgoal is therefore mapped to a belief base entry, which contains its name and a value that may be changed by the user at run-time. Such a value expresses the actual softgoal impor-

[2]Jade, as well as Jadex, are based on a pure Java API. More details at http://jade.tilab.com/ and http://vsis-www.informatik.uni-hamburg.de/projects/jadex/

tance and may change from time to time reflecting environmental changes.

Plan. This mapping considers only those *Tropos* plans that have a direct means-end relationship to leaf goals, namely root-level plans according to our definition of capability concept.

Resource. Resources naturally map to an entry or a set of entries in the Jadex belief base. Since in Jadex the belief base is an object-oriented database, the entry can be related to an arbitrary Java object. At runtime, resources should be changeable by means of an external request message, to reflect changes in the environment.

In order to endow the generated BDI agents with all the *GM* features, We provide the mappings for a wide subset of *Tropos* relationships into Jadex structures. Below, for space reasons, we give just few details.

AND decomposition. If an AND-decomposed goal is activated, all subgoals have to be dispatched. Specifically, an AND-decomposed goal is set as trigger for exactly one plan, called AND-dispatch-plan. In the plan body, all subgoals have to be dispatched in (some, perhaps random) sequence, if one subgoal fails, the process has to be stopped and a failure has to be returned. For this first proposal, on failure no attempts for compensation techniques of already executed actions have been considered.

OR decomposition. Jadex goals cannot activate other goals, but only be the triggering event for a plan. So, to map this kind of decomposition, Jadex-plans have to be linked between goals and the OR-decomposed sub-goals, as illustrated in Figure 2. Each dispatch-goal plan (hexagon) is triggered on the activation of the parent goal and it dispatches one subgoal. Since an OR-decomposition deals with at least two goals (and related plans) as alternative ways to achieve the triggered goal, the agent needs to be able to reason about what is the more convenient at that time. To deliver on such a task, as shown in in Figure 2, we adopted the Jadex meta-level reasoning. That is, if more than one plan is applicable for an active goal, such a meta-reasoning process starts: a so-called *metagoal* is dispatched, which triggers an associated plan, the *metaplan*, that implements a strategy (e.g. some AI techniques) to select between applicable plans.

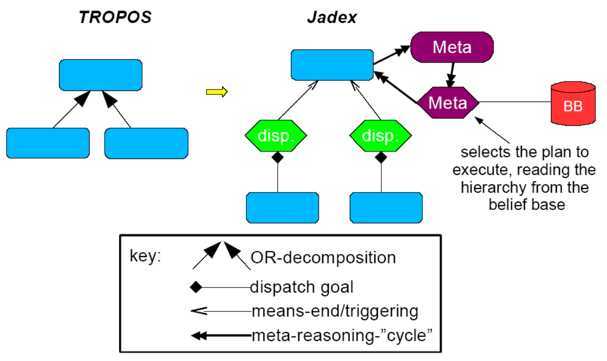

Figure 2: Mapping of the *Tropos* goal OR-decomposition into an equivalent Jadex BDI structure.

Means-end. The *Tropos* means-end relationship can be mapped one-to-one to the Jadex plan triggering mechanism. Having defined no conditions, every time the associated goal is activated, plan execution is triggered. Notice that, in this case, the Jadex plans are real *Tropos* root-level plans, namely those required to build up agent capabilities. Jadex supposes that every applicable plan for a goal (without achievement conditions) is able to satisfy that goal completely. Therefore, if more than one plan is applicable, a meta-level reasoning is utilized in the same way seen for the goal OR-decomposition in Figure 2.

The generated agent can evaluate costs for the execution of every plan in order to effectively deal with goals and plans selection. Costs include softgoal contribution and importance; at run-time, negative contributions could cause higher cost and guide the agent to the selection of alternatives for goal satisfaction.

4. CONCLUSION AND FUTURE WORK

This paper presented a tool-supported, systematic process for generating BDI agents from goal models. We described an agent design framework that is flexible enough to support proactive, deliberative and reactive agents without focusing on domain specific AI techniques. The generated agents are able to reason about their intentionality, being aware of their potential behaviors along with associated events and environmental constraints.

Our future work will extend the proposed framework providing automatic execution of experimental tests. We also intend to refine the framework to support the design of composite/organizational agents which consist of an aggregation hierarchy.

5. REFERENCES

[1] 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.

[2] L. Cernuzzi and F. Zambonelli. Dealing with Adaptive Multi-Agent Organizations in the Gaia Methodology. In *6th International Workshop on Agent-Oriented Software Engineering (AOSE-2005)*, 2005.

[3] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. In *6IWSSD: Selected Papers of the Sixth International Workshop on Software Specification and Design*, pages 3–50. Elsevier Science Publishers B. V., 1993.

[4] E. Durfee and T. Montgomery. Coordination as distributed search in a hierarchical behavior space. In *Systems, Man and Cybernetics, IEEE Transactions on*, volume 21. 1991.

[5] B. Henderson-Sellers and P. Giorgini, editors. *Agent-Oriented Methodologies*. Idea Group Inc., 2005.

[6] N. Jennings. *Foundations of Distributed Artificial Intelligence*, chapter Coordination Techniques for Distributed Artificial Intelligence. Wiley-IEEE, 1996.

[7] V. Lesser. A retrospective view of fa/c distributed problem solving. In *Systems, Man and Cybernetics, IEEE Transactions on*, volume 21. 1991.

[8] L. Penserini, A. Perini, A. Susi, M. Morandini, and J. Mylopoulos. A Design Framework for Generating BDI-Agents from Goal Models. (Extended version). Technical report, ITC-irst, 2007.

[9] L. Penserini, A. Perini, A. Susi, and J. Mylopoulos. From Stakeholder Intentions to Software Agent Implementations. In *Proceedings of the 18th Conference On Advanced Information Systems Engineering*, volume 4001 of *LNCS*, pages 465–479. Springer-Verlag, 2006.

[10] E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, Department of Computer Science, University of Toronto, 1995.