A Design Framework for Generating BDI-Agents from Goal Models
Technical Report ITC-IRST, October 2006

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
Goal models have been proposed in Distributed Artificial Intelligence for guiding software agents at run-time. In these approaches goal models consist of goal graphs representing AND/OR hierarchical decomposition along with inter-dependency links representing conflicts between goals or resources needed to satisfy them.

Goal-oriented software requirements engineering exploits similar goal structures to support human analysts, at design-time, in analyzing the requirements of a system-to-be.

We propose the use of goal models at different abstraction levels in engineering a Multi-Agent System (MAS), that is, not only at design time, but also as a part of the agent knowledge and choice strategy, at run-time. This approach aims at addressing crucial issues in complex distributed software such as evolvability and adaptivity.

We define a tool-supported design framework that allows to specify an agent goal model and to automatically generate from it fragments of a BDI agent. We devise the design process as a transformation process from design, platform-independent models, to platform-specific models and then to code, following Model Driven Architecture ideas. The design framework is demonstrated by referring to the Tropos methodology and to the JADE/Jadex MAS platform. The process is illustrated through an example and experimental results on various goal models are discussed.

1. INTRODUCTION
Goal models have been used in Distributed Artificial Intelligence as a means for capturing agent intentions and guiding agent coordination [8, 13, 12]. 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 [6, 23, 3]. 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, GMs provide analysis and design artifacts during system development. GMs 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 [6, 9].

Taking advantages of the above results, we propose to use GMs 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 explicit the dependencies between system agent goals and stakeholder goals. 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 system behavior can be obtained by querying a design-time GM. For instance, considering a MAS supporting different word searching over the internet, we may query our design-time GM to determine whether the system can manage a request (event) by a user. For instance, finding grammatical/semantic information about a word might be accomplished by either searching with 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 GMs and generates fragments of a BDI agent. These fragments include goals and capabilities, along with a reasoning strategy for selecting and running appropriate capabilities for a given goal and domain conditions.

Our approach offers a systematic process for operationalizing a GM into a set of capabilities. We automate BDI-agent code generation from the GM design artefact, and evaluate experimentally the fact that the resulting agents are enabled to reason about their intentionalities, namely they are aware of their potential behaviors along with associated events and environmental constraints.

This approach aims at addressing crucial issues in developing and maintaining complex distributed software which motivated the definition of frameworks and guidelines such as the Model Driven Architecture (MDA) framework [14]. MDA calls for devising a system development process as a transformation process of different abstraction level models: from design, platform-independent models, to platform-specific models and then to code. Model query,
as well as transformation and traceability mechanisms from one level to the other are also requested. A few Agent-Oriented Software Engineering (AOSE) methodologies are exploiting MDA concepts [11], but, up to our knowledge, none of them provides a structured and tool supported process to transform knowledge-level design artifacts into execution time artifacts. Moreover, the use of GM offers an interesting direction for building adaptive systems, taking sign artifacts into execution time artifacts. Moreover, the use of GM is used also to determine different agent types (roles). Despite different methodologies and techniques, the late design and the implementation phases mainly rely on the use of the goal model in order to figure out system requirements. Nevertheless, the output of the analysis and design process with Gaia corresponds to an abstract specification that requires to be operationalized by using additional detailed design methods and techniques.

In [16] the Prometheus methodology makes use of goal models to describe system requirements. Analogously to [3], after a goal model is built, the designer identify those goals that are related to system functionalities (by the use of descriptors) and delegate them to specific system actors. Then functionalities are grouped in order to characterize scenarios, namely sequence of steps (functionality) in order to achieve a goal. Notice that, this grouping mechanism is used also to determine different agent types (roles). Despite the use of the goal model in order to figure out system requirements, the late design and the implementation phases mainly rely on the system overview diagrams that do not relate with previous goal model elements and properties. Instead, they focus on agent capabilities and events that may trigger them. This also reflects in the final agent code. That is agent awareness about its goal model is limited. For example, designed agent behavior is mainly reactive rather than proactive and deliberative; the agent cannot automatically reason on its goal model in order to deal with failures and to choose alternative behaviors. Moreover, also traceability from/to design artifacts is not supported.

Hermes [5] aims at overcoming weaknesses of the above mentioned methodology, considering the goal model a core element of the implemented agent. In Hermes the generated BDI agent is aware of its goal model, named Interaction Goal Hierarchy Diagram, which is used to characterize behavioral strategies to cope with social commitments. This gives a more flexible approach respect to traditional message-centric agent interaction approaches. This methodology needs further research in order to deal with a complete design framework, for instance it does not cover the requirements analysis and architectural design phases.

The work presented in [7] is relevant to our research, even if it adopts a different set of concepts. This work focuses on business interaction protocols, defined according to an agent-oriented paradigm. Agent commitments may trigger business interaction protocols. A protocol models possible ways to adapt to a variety of changes. In our opinion, agent commitment corresponds to our concept of agent goal and protocol to our agent behavior. Moreover, agent roles are defined in terms of local processes (a concept analogous to our capability) that can be composed to define more complex business process involving several agents. Despite the authors claim that commitments capture a variety of contractual relationships, while enabling manipulations such as delegation and assignment, an explicit goal model to represent them is missing.

2. RELATED WORK

Different Agent-Oriented Software Engineering methodologies use similar goal model concepts to analyze system requirements or to specify agents [5, 16, 24, 3].

Gaia, one of the most prominent AOSE methodologies [24], exploits the so called preliminary role model in the analysis phase. This model implicitly carries out information about agent goals corresponding to system requirements. Nevertheless, the output of the analysis and design process with Gaia corresponds to an abstract specification that requires to be operationalized by using additional detailed design methods and techniques.

In [16] the Prometheus methodology makes use of goal models to describe system requirements. Analogously to [3], after a goal model is built, the designer identify those goals that are related to system functionalities (by the use of descriptors) and delegate them to specific system actors. Then functionalities are grouped in order to characterize scenarios, namely sequence of steps (functionality) in order to achieve a goal. Notice that, this grouping mechanism is used also to determine different agent types (roles). Despite the use of the goal model in order to figure out system requirements, the late design and the implementation phases mainly rely on the system overview diagrams that do not relate with previous goal model elements and properties. Instead, they focus on agent capabilities and events that may trigger them. This also reflects in the final agent code. That is agent awareness about its goal model is limited. For example, designed agent behavior is mainly reactive rather than proactive and deliberative; the agent cannot automatically reason on its goal model in order to deal with failures and to choose alternative behaviors. Moreover, also traceability from/to design artifacts is not supported.

Hermes [5] aims at overcoming weaknesses of the above mentioned methodology, considering the goal model a core element of the implemented agent. In Hermes the generated BDI agent is aware of its goal model, named Interaction Goal Hierarchy Diagram, which is used to characterize behavioral strategies to cope with social commitments. This gives a more flexible approach respect to traditional message-centric agent interaction approaches. This methodology needs further research in order to deal with a complete design framework, for instance it does not cover the requirements analysis and architectural design phases.

The work presented in [7] is relevant to our research, even if it adopts a different set of concepts. This work focuses on business interaction protocols, defined according to an agent-oriented paradigm. Agent commitments may trigger business interaction protocols. A protocol models possible ways to adapt to a variety of changes. In our opinion, agent commitment corresponds to our concept of agent goal and protocol to our agent behavior. Moreover, agent roles are defined in terms of local processes (a concept analogous to our capability) that can be composed to define more complex business process involving several agents. Despite the authors claim that commitments capture a variety of contractual relationships, while enabling manipulations such as delegation and assignment, an explicit goal model to represent them is missing.

3. CONCEPTUAL FRAMEWORK

As with other approaches, in our framework a GM is a goal graph consisting of a forest of AND/OR goal trees, along with inter-dependency links between goals and means-end relationships between leaf-level goals and plans that represent a way to achieve these goals. The resulting GM provides also a schema of the possible behaviors an agent can use to fulfill its goals.

We define capability the sub-graph rooted in a leaf goal containing the set of means-end plans with their inter-dependency relationships towards other goals. We call knowledge-level design the process of building the higher level part of a GM and capability-level design the process of refining leaf goals into plan means-end and inter-dependency relationships. This last design step represents a way to operationalize goals, that is to define the possible behaviours of an agent.

More formally, called BS, behavior-schema, the set of all possible behaviors Bh an agent can play, E the set of events\(^1\) an agent can perceive, C the set of constraints (e.g. user preferences and system QoS), G the set of goals an agent can achieve, and \(Cp\) a set of capabilities an agent can exploit. Be \(G \subseteq G\), the set of leaf-level goals an agent can operationalize. Adopting all these elements, we define agent behaviors and two functions to query respectively the behavior-schema and a GM structure.

\[\text{behavior-schema} \quad BS = \{Bh_1(X_1), \ldots, Bh_n(X_n)\} \], where \(Bh_i\) are sub-schema representing a possible BS decomposition. Each \(X_i\) is a list of attributes of several type: \(X_i = \{\text{Events, Constraints, Goals}\}\) where Events assumes values in \(2^{E}\); Constraints assumes values in \(2^{C}\); Goals assumes values in \(2^{G}\) that characterize the sub-schema behavior \(Bh_i\).

\[\text{behavior-schema function} \quad \text{For each } BS \text{ exists a behavior-schema function that univocally associates a set of events and constraints to a set of leaf-level goals. That is: } f_{Bh} : 2^{E} \times 2^{C} \rightarrow 2^{G} \text{ allows an agent to come out with the right } Bh(X) \text{ when an event occurs and some constraints are perceived.}\]

\[\text{capability function} \quad \text{For each behavior schema } Bh \text{ (sub-schema) exists a capability function such that: } f_{C} : 2^{E} \rightarrow 2^{C} \text{. Once, for a } Bh(X) \text{ the right capabilities have been selected, the agent can actually deal with a behavior instance } b \in Bh(X).}\]

The behavior-schema and capability functions allow to query about agent properties with reference to concrete instances of behaviors and capabilities, each time an event occurs and constraints are sensed. For example, each time an agent receives a request message (an event), it interprets it in order to extract the goals to

\(^1\)Examples are messages that will trigger a goal.
be triggered, and concurrently perceives environmental conditions \((C)\) to better choose the right behavior. As a composition of the previous ones, we have also the following agent property:

**behavior-instance function**

The previous \(f_{bh}\) can be composed with the \(f_{cp}\) as follows:

\[
F_b = f_{bh} \circ f_{cp} : 2^F \times 2^G \times 2^{E_1} \rightarrow 2^{E_0}.
\]

This behavior-instance function definition indicates that an agent behavior is always associated to a set of capabilities, but while the agent behavior can be observed at run-time (even in an unpredictable way), the capability is a design-time concept.

A unifying abstraction that integrates all the above defined properties is that of agent role, defined in the following, according to the previously given definitions:

**agent role**

An agent \((ag)\) can play several roles, which are characterized by different objectives (in respect how to perceive and to react to the environmental stimulus), user expectations (e.g. preferences and QoS), and capabilities. Therefore, according to the previous defined concepts, an agent role (Role) is an element structure defined as follows:

\[
\text{Role} = (Ag, BS, 2^E, 2^G, 2^{E_1}, 2^{E_0})
\]

where \(Ag\) is a set of agents that are currently playing the Role and for the sets \(2^E, 2^G, 2^{E_1}, 2^{E_0}\) only the set of values in \(BS\) are considered.

Similarities exist among our formal description and the database theory [1]. Our agent role knows about its possible behaviors, as a Database embeds a set of relations (i.e. schema of relations), and similar analogies can be drawn between the concept of Behavior Schema \(BS\) and that of Database \(DB\) schema, and between the concept of agent behaviors and that of relations in a Database schema.

We exploit this conceptual framework during MAS development and instantiate it using the Tropos methodology for analysis and design, and the JADE/Jadex agent platform for the implementation.

### 4. FROM GM DESIGN IN TROPOS TO JADEX-JADE BDI-AGENTS

#### 4.1 Concepts

The Tropos agent-oriented methodology [3] borrows modelling and analysis techniques from goal-oriented requirements engineering frameworks and integrates them into an agent-oriented paradigm.

The development process, in Tropos, is organized into five phases: *Early Requirements* whose objective is to produce a model of the environment (i.e. the organizational setting); *Late Requirements*, in which the system-to-be is introduced in the domain and its impact within the environment is analyzed; *Architectural Design* whose objective is to obtain a representation of the internal architecture of the system, in terms of subcomponents of the system and relationships among them; *Detailed Design* which is concerned with the definition of software agents rationale, including capabilities and interactions specifications; *Implementation*, whose objective is the production of code from the detailed design specification, according to an established mapping between the implementation platform constructs and the detailed design elements.

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; supporting tools [20].

Adopting Tropos into our framework allows us to represent and reason on GM resulting from the analysis of each actor point of view. More 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 \(g_1,+,+\) and \(g_2\) 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 illustrate a fragment of a goal-oriented Tropos specification of the requirements of a search system. The system is intended to support students and teachers in exam related activities. 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). The models includes the two main domain stakeholders (Student, Teacher) along with their main goals (e.g. pass exam, find word description, find copied text), and mutual dependencies for goal achievement.

The balloon associated to the system actor, Search Actor, depicts a goal analysis conducted from the point of view of that actor. The goal find copied text is AND decomposed into the sub-goals find word description and find matching. Sub-goals are further refined into more concrete goals till a plan is found that provides a means to achieve the goal. Inter-dependency links between goal models associated to different actors can be identified along actor dependency links. For instance between the GM of the actor Search Actor and The GM of the actor Exam Parser, along the goal dependency centered on the goal Parse exam.

For the concept of capability, we adopt the revised definition proposed in [19], 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 it is specified in Tropos by a means-end relationship between the goal and the plan. For example, in Fig. 1, the plan Encyclopedia in means-ends relationship with the goal find in encyclopedia corresponds to the ability part of a capability. The opportunity component represents user preferences and environmental conditions, which may enable or disable the execution of the ability component, at run time. It is represented in terms of plan/softgoal contributions, \((\text{plan}, \text{softgoal}, \text{metric}) (\text{metric} \in \{-,−,+,+,++\})\), and environmental constraints (e.g. temporal constraints between sub-plans) which can be specified by model annotations. The contribution link between the plan Encyclopedia and the softgoal reliable results is part of the opportunity specification for the capability identified by the leaf goal find in encyclopedia.

Different means (plans), to achieve a goal, model alternative capabilities. As an example, in Fig. 1, the goal find in Encyclopedia and the plan Wikipedia define a capability (i.e. \(cp_1\)) and the same goal with the plan Encyclopedia define another capability (i.e. \(cp_2\)).

More formally, the definition of capability is given by:

\[
 cp = (\text{means-end}(\text{goal}, \text{plan}), \\
 \cup_{\text{contribution}(\text{plan}, \text{softgoal}, \text{metric}),} \{AN_1, \ldots, AN_n\})
\]

where \(\text{contribution}(\text{plan}, \text{softgoal}, \text{metric})\) is the set of contribution relationships of the plan \(\text{plan}\) to the softgoals \(\text{softgoal}\), —according to a specific metric \(\text{metric}\) — and \(\{AN_1, \ldots, AN_n\}\) is a set of model annotations describing domain constraints. Annotations can be specified by the Formal Tropos language [10], a first

---

2 For more details on modelling activities see [3].
order temporal logic language, to specify constraints on the model elements as annotations. This goal-oriented specification is complemented by a representation of the dynamic properties of a capability in UML notation, namely UML2.0 activities and sequence diagrams. In this approach, each root-level plan may be described by an activity diagram, where leaf-level plans are modelled as activities. Indeed, the Tropos plan AND/OR decomposition comes out with leaf-level plans suitable to model atomic agent actions.

In the rest of this subsection we will describe our systematic process for operationalizing a GM into a set of capabilities, using the example depicted in Figure 1.

In particular, we will point out the two different abstraction levels that characterize the agent design, namely the knowledge level and the capability level. The knowledge level, refers to the goal AND/OR decomposition (part of the GM) that contributes to the description of the behaviors the specific agent role can play. In other words, this part of the GM 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 and 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. This represents an example of a behavior sub-schema (BH(X)), as defined in Section 3.

Under this modelling properties, we have assumed, for our prototype, that inside a GM only goals having an in dependency link can be triggered by external message events. As a further example, let us consider the goal dependency find the word description, between the actor Student and the agent role Search Actor, that models the possibility for such a goal to be triggered by a request message from the user (or a personal agent of Student). Therefore, some of GM’s goals are known by external actors, while others are private in the sense that indicate agent internal strategies never publicized to solve problems.

![Figure 1: Tropos architectural design: Agent knowledge and capability levels.](image)

A GM represents the agent intentionalities in terms of how the agent perceives the environment, applies strategies to fulfill its responsibilities, and chooses alternative ways to adapt to requirements changes.

Nevertheless, as shown in Figure 1, the more operative part that the agent requires to actually affect the environment is specified in the capability level. In other words, like a human being, the agent perceives the environment and thinks to a suitable behavior to be accomplished (knowledge-level), but to actually adopt such a behavior it needs to execute specific capabilities (capability-level). Looking at the Figure 1, if the goal find the word description is triggered, as this is OR-decomposed in the two sub-goals, in this case, the agent role Search Actor has two possible behaviors to satisfy the initial triggered goal: one that brings about the goal find in Internet satisfaction and the second that brings about the goal find in encyclopedia satisfaction. Specifically, in our agent goal analysis, leaf node goals are satisfied by root-level plans through the Tropos means-end relationship. As already said, this plans univocally identify the capabilities the agent can play.

### 4.2 Design process

The systematic software development process we propose takes advantages of ideas and standards specified by the Model Driven Architecture (MDA) initiative [14]. MDA conceives system development in terms of a chain of model transformations, namely, from a domain model (Computationally Independent Model — CIM) to a Platform Independent Model (PIM), and from a PIM model to a Platform Specific Model (PSM). From PSM code and other development artifacts can then be derived straightforwardly.

A CIM model is built applying knowledge and capability level modeling. While the first one corresponds to the traditional Tropos analysis and modeling activities, as described, for instance, in [3], here we briefly recall the capability modelling process, that has been firstly introduced in [19, 18].

The capability design process is composed by two main steps, as follows:

**Step 1.** Capability modeling starts during requirements analysis by identifying agent capabilities and their correlations with stakeholder needs. Both, ability part and opportunity part are modeled and are collected in a capability table, as the one in Tab 1. Tab 1 depicts all the capabilities that operationalize the knowledge level of GM of the Search Actor in the example given in Fig. 1.

<table>
<thead>
<tr>
<th>Capabilities</th>
<th>Means_End/Goal/Plan</th>
<th>List of Contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td>cp1</td>
<td>filter pages,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>contentFilter</td>
<td></td>
</tr>
<tr>
<td>cp2</td>
<td>parse result,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>parser</td>
<td></td>
</tr>
<tr>
<td>cp3</td>
<td>find in encyclopedia,</td>
<td>reliable result +</td>
</tr>
<tr>
<td></td>
<td>Wikipedia</td>
<td></td>
</tr>
<tr>
<td>cp4</td>
<td>find in encyclopedia,</td>
<td>minimize cost +</td>
</tr>
<tr>
<td></td>
<td>EBritannica</td>
<td></td>
</tr>
<tr>
<td></td>
<td>minimize cost -</td>
<td></td>
</tr>
<tr>
<td>cp5</td>
<td>find matching,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>matchText</td>
<td></td>
</tr>
</tbody>
</table>

**Table 1: Search Actor capabilities.**

At this stage, we deal with dynamic properties of the capability by transforming Tropos modelling concepts (CIM) both into UML activity diagram concepts (PIM) for the ability part and into BDI agent concepts (PIM) for the opportunity part. That is, in the case of the ability, generated activity diagrams are enriched with sequence diagrams to specify execution workflow and interaction protocols respectively [19]. While, for the opportunity, Tropos softgoal contribution relationships are mapped to generic BDI agent struc-
tures according to the Agent Definition File specifications proposed by the Jadex development agent platform [21].

**Step 2.** Agent code generation results from a transformation of the previous (step 1) PIM specification to the target platform specific model (PSM), i.e., Jade agents in our case. Specifically, for the ability part, PIM to PSM transformation is implemented by using model driven automatic transformation techniques (as detailed in [19]); while, for the opportunity part, the second step relies on an automatic transformation from Tropos models to the Jadex/Jade adapter [21], as detailed in next sections.

The output of these two steps based process consists of a skeleton of the DBI agent along with its capabilities implementation.

### 4.3 Towards Implementation

As already said, the GM design artifact is composed of the knowledge-level and the capability-level parts. While the capability level design and implementation have been already described in [19, 18], in this Section, we deal with the agent knowledge-level implementation. We describe the semantic of the adopted sub-set of Tropos concepts, and we give an overview of some of the proposed mappings between such concepts and related data-structures of a Jadex BDI agent. That is, we go from a platform independent model (PIM) to a platform specific model (PSM). The proposed approach considers Tropos agent GMs, from the architectural design phase, as the PIM model; while, the proposed agent implementation naturally accommodates with a BDI-architecture that is built on the Jadex platform (i.e. PSM).

Therefore, the specification for the mapping 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.** Softgoals are considered as abstract entities more related to beliefs and desires than to goals and plans. For example, in our prototype, they are mainly used to define opportunities for the selection of the next goals or plans to pursue along the GM. That is, they may model domain constraints \(c \in C\) to drive the selection of the more convenient behavior (both \(Bh(X)\) once an event \(e \in E\) occurs. A softgoal is therefore mapped only 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 softgoal actual importance and may change 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.

Tropos allows designer to adopt its concepts and diagrammatic notation along all the development phases, while our effort has been to select a sub-set of them suitable to be used at architectural design in order to correctly deal with the agent design and implement-

---

5Jade, as well as Jadex, is based on a pure Java API. More details at: http://jade.tilab.com/

**AND decomposition.** If an AND-decomposed goal is activated, all subgoals have to be dispatched. As illustrated in left-hand side of Figure 2, the following Jadex solution was adopted: an AND-decomposed goal is set as trigger for exactly one plan, called AND-dispatch-plan (green hexagon). 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.** As previously seen, in 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 3. 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 3, 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 metaplan 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 AND-decomposition into an equivalent Jadex BDI structure.

Figure 3: 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 3.

delegation. We consider this as the pure Tropos delegation relationship between the user and the software agent, where the user makes requests to the system to satisfy a goal. Therefore, the system has to be able to handle external goal satisfaction requests. For example, looking at Figure 1, Student and Teacher respectively delegate goals find word description and find copied text to the agent role Search Actor. In Jadex, to each delegated goal a plan, called request-plan, is associated, that can be triggered by a request message for that specific goal. This plan has to handle the standard FIPA-Request interaction protocol, informing the user for acceptance or rejection of the request, and it has to dispatch that goal and to return success or failure information regarding goal achievement.

Figure 4: Mapping of the Tropos why-dependency into an equivalent Jadex BDI structure.

‘why’-dependency. The Tropos why-dependency relationship between two agents can be easily mapped to a FIPA-Request interaction protocol, like a delegation. Figure 4 shows that the dependent agent has to make a request to the dependee, as this request carries out the information about goals to be achieved. To the Jadex side, a special service plan has to be associated to the goal that needs the dependency. This plan has the role of initiator for the request protocol; while, its counterpart, the dependee, can be realized implementing the same request-plan seen for the delegation. In the case the (dependum) goal cannot be achieved, the dependee agent has to return a failure.

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; namely, negative contributions could cause higher cost. They can be requested to other agents, summed, and transparently modified by the programmer. Moreover, each agent endows the knowledge about its GM goal relationships (i.e. AND/OR decompositions, dependencies, delegations, and contribution links) into its belief base. For the sake of brevity, this paper does not detail such a part.

5. EXPERIMENTAL TESTS
As the Tropos methodology is tool-supported, we have extended the TAOM4E development environment [19], with the mapping features from Tropos to Jadex. Figure 5 shows the principal parts of the tool front-end. The largest windows, on the right hand-side, supports models editing according to the Tropos graphical notation that is provided by the Palette window (in the center). Notice that, visualized diagrams are often partial views on the whole model.

For the Tropos to Jadex automatic BDI-Agent code generation, the interesting part is shown in the left-hand side windows, titled Navigator. Here, the Tropos project Searcher.tropos contains all the design diagrams as the one illustrated in the editing window. This latter shows a multi-agent system Search System that is composed of three agent roles: Web Server, Search Actor, and Exam Parser. The window Navigator also embeds the generated code for the three BDI-agents, as illustrated by the folders contained in the project folder t2x. Such generated agent roles are self-contained, namely ready to be launched and tested.

In details, at Tropos architectural design phase, each agent has a balloon (dash line ellipse) that describes the agent rational in terms of knowledge and capability levels. Notice that, Tropos notation allows to correlate stakeholder needs with agent internal capabilities. For example, Student has the need to look for word meaning on the Web (i.e. goal find word description); in order to fulfill this goal she depends on the agent role Search Actor capabilities. Again, Teacher’s needs are more related to provide students with a reliable service at a minimum cost, so she depends on the same agent for softgoals reliable result and minimize cost satisfaction. Preliminary experiments consist in monitoring our generated agents by Jadex Control Center Tracer, verifying under specific conditions that it behaves as previously designed.

Example 1. Let us assume that Student asks to the agent Search Actor to seek for a specific word description, over the Internet. According to the Search Actor GM shown in Fig. 1, and without considering any kind of failure, a possible agent behavior could be as the one described in the following. We describe such a scenario by means of the provided formalization related to agent behavior-schema. Specifically, an agent plays the role of Search Actor and receives, from a user that is playing the role Student, a request-message event (e ∈ E) that implicitly carries out information about what goals have to be achieved, i.e. the goal find word description ∈ G. The agent Search Actor also perceives some context conditions modelled as softgoals: (minimize cost, reliable results) ∈ C. Moreover, Search Actor can apply the function BH in order to select the more appropriate behavior-schema at the time the event occurs, namely it decides what leaf-level goals need to be achieved. In our case, the softgoals contribution links drive the selection towards the leaf-goal find in encyclopedia ∈ GL. At this time, the agent has identified all the attributes (X) that characterize a behavior-schema BH(X). But, in order to actually exploit a behavior, the agent needs to create an instance b ∈ BH(X). As a behavior instance b is operatively composed of a set of capabilities to be executed, the agent takes advantage of its fc+p property. Turning to our example, as such a leaf-goal can be achieved by two capabilities (cp1 and cp4), our agent has to apply the function fc+p to select the more appropriate capability: the best choice, in terms of contribution links, is given by cp3.

Example 2. While considering the Example 1, let us assume that both the web sites related to the plans Wikipedia and EB Britannica are down when the request occurs, i.e. in such a case the agent Search Actor capabilities cp1 and cp4 fail. This scenario is
illustrated in Figure 6 that also gives an idea of what kind of experiments we can execute exploiting the prototype. At the time the request occurs, from the observed agent Search Actor (right-side of Figure 6), several behavioral branches are generated and monitored by the Jadex Control Center Tracer. As illustrated by the hand-made links between the Tropos behavior-schema design (left-side of Figure 6) and the Jadex monitored agent behavior instance (on the right in Figure 6), some of these branches represent alternative ways to achieve goals that failed (as in the case of the plans Wikipedia or EBritannica) and others that bring the agent to the success (plans contentFilter and parser). Notice that, when the goal find in Internet has to be achieved, according to the why-link dependency, the Search Actor makes a request to the Web Server (that for lack of space is not shown in Figure 6) in order to ask for the fulfillment of the goal find webpage, i.e. the branch labeled Dependency in the right part of Figure 6. Obviously, only in the case such a request returns a success, the Search Actor continues with the execution of its internal plans, contentFilter and parser, required to achieve the initial goal find in Internet.

In our experimental tests, the generated BDI-agent prioritizes alternatives considering the weights associated to the contribution links, when exist, otherwise it executes a random-based selection. Moreover, the strict correspondence between Tropos design and Jadex run-time, allow a designer to verify, currently only by inspection, that the generated agent behaves as designed. This does not mean that from the Tropos agent GM, the designer can predict the behavior instance. On the contrary, the implemented GM only gives to the generated agent the flexibility to reason about alternatives to achieve goals, but subsequently such an agent could adopt any criteria to select one of them, e.g. the use of some AI technique.
6. CONCLUSIONS AND FUTURE WORK

This paper presented a tool-supported, systematic process for generating BDI agents from goal models. Specifically, we described an agent design framework that is flexible enough to support the most important type of agents: proactive, deliberative and reactive without focusing on domain specific AI techniques. The generated agents are able to reason about their intentionalities, namely they are aware of their potential behaviors along with associated events and environmental constraints. The design process has been illustrated through an example and tested with several goal models.

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.

7. REFERENCES