From Capability Specifications to Code
for Multi-Agent Software

Loris Penserini, Anna Perini, Angelo Susi
ITC-IRST
Via Sommarive 18
I-38050, Trento, Italy
{penserini,perini,susi}@itc.it

John Mylopoulos
University of Trento
Via Sommarive, 14
I-38050, Trento, Italy
jm@cs.toronto.edu

Abstract

Current ICT application domains, such as web services and autonomic computing, call for highly flexible systems, capable of adapting to changing operational environments as well as to user needs. Multi-agent system framework do include mechanisms that make flexibility and adaptability possible. In our research we focus on how to take into account environmental constraints and stakeholder needs in the design of software agent capabilities.

1. Introduction

The advent of global connectivity makes computing resources ubiquitous (to anyone, anytime, anyplace). In such settings, software requirements evolve along with changing environments and user needs. This calls for software that is flexible and adaptive.

To design such software, we adopt multi-agent system (hereafter MAS) concepts to specify requirements and designs. Specifically, we work within the framework of the Tropos methodology [1, 3], for building agent-oriented software systems. Our work exploits ideas and standards from the Model Driven Architecture (MDA) initiative [2] related to model to model transformations. In particular, our framework includes a first transformation from Tropos model (CIM - Computational Independent Model) to both AUML and beliefs-desires-intentions [9] agents (PIM - Platform Independent Model), then a second transformation from the PIM models to Jade-based coding (PSM - Platform Specific Model).

The main objective of this paper is to revisit the notion of agent capability in Tropos to align it with recent proposals [4]. These argue convincingly that capability presumes both the concept of ability and the concept of opportunity, respectively representing the necessary and the sufficient conditions for achieving a goal.

The paper is structured as follows. In Section 2, we recall basic elements of the Tropos methodology and present the revised definition of agent capability, then we give an overview of our development process. Section 3 deals with capability design and implementation, focusing on the opportunity part. Finally, conclusions are drawn in Section 5.

2. Background

The Tropos agent-oriented methodology [1] is aimed at supporting the whole development process, from early requirements to implementation. It provides a conceptual modeling language, based on concepts of actor, goal, plan, resource, and of social dependency between actors for goal achievement; analysis techniques; tool support [7].

The Tropos development process 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. Notice that, in this short paper we will only focus on the detailed design and implementation phases.

The framework provides a graphical notation to depict views of a model, namely, actor diagrams, which allows to illustrate dependencies between a set of actors and goal diagrams, which allows to view results of goals analysis from an actor’s point of view (examples will be given in
This work aims at adopting a revised definition of the agent capability notion in Tropos, which distinguishes the concept of ability from the concept of opportunity [5], in order to endow the capability with stakeholder needs till down its implementation. The ability part represents a way to achieve a given goal. The opportunity part represents user preferences and environmental conditions, which may enable or disable the execution of the ability part, at run time. Moreover, we propose to anticipate capability modeling, since the Tropos requirements analysis phases, providing guidelines and tools to support the design process. In particular, 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. The ability part of capabilities is specified in Tropos by a means-end relationship between the goal and the plan. Hence, several means (plans), to achieve a goal, model alternative capabilities. As an example, in Fig. 1, the goal provide product info and the plan browse catalog define the ability part of a capability (i.e. \( \text{Cap} \_1 \)) and the same goal with the plan solve query the ability of another one (i.e. \( \text{Cap} \_2 \)). While, the opportunity part may represent stakeholder needs in terms of preferences and QoS, which may enable or disable the execution of the ability part. It has been described in Tropos via plan/softgoal contribution relationships, e.g. in Fig. 1, the plan solve query gives a strong positive contribution (‘++’) to the softgoal fast search on catalogs satisfaction.

At this stage, we deal with dynamic properties of the capability by transforming Tropos modelling concepts (CIM) both into AUML activity diagram concepts (PIM), i.e. Jade\(^1\) agents in our case. Specifically, for the ability part, PIM to PSM transformation is implemented by using automatic transformation techniques (as detailed in [5]); while, for the opportunity part, the second step relies on the Jadex/Jade adapter [8] already included into the Jadex tool.

The output of these two steps based process consists of a skeleton of the agent capability implementation. Specifically, the complete agent capability implementation is given as BDI structures in Jadex: the ability part correspond to a plan to achieve a goal and the opportunity part deals with agent beliefs and goals.

![Figure 1. Detailed Design: fragment of capability design of the agent Search Manager.](image)

3. Dealing with Agent Capability

According to our approach, this Section focuses on the opportunity part of the agent capability, referring to an example from the on-line selling scenario partially illustrated in Fig.1. While for the ability part of the design and the implementation, more details are given in [5].

### 3.1 From stakeholder needs to implementation

Concerning the opportunity part of a capability, we conceived its transformation process from a Tropos conceptual model (CIM) to a BDI agent model (PIM). Specifically, our BDI agent model is based on the Jadex framework that aims at supporting a platform independent model for BDI agents development; namely, Jadex agents can potentially run on any platform that provides basic agent management and messaging functions. Currently, middleware adapters for Jadex have been only realized for the agent platform Jade [8]. In particular, Jadex abstracts from the specific model by means of the so called agent definition file (ADF), which carries out the complete definition of a BDI agent.

\(^1\)Jade, as well as Jadex, is based on a pure Java API. More details at: http://jade.tilab.com/
During capability design, stakeholder needs are correlated to agent capabilities (via their opportunity part) by means of the Tropos contribution relationships analysis. For example, let’s consider the proposed scenario for capabilities $\text{Cap}_1$ and $\text{Cap}_2$, as detailed in Fig. 1. Specifically, their ability parts —i.e. plans browse catalog and solve query— may affect with different strength the achievement of specific softgoals as follows:

**search for desired product automatically.** This softgoal models a Customer need, meaning that each time the agent $\text{Search Manager}$ profiles the Customer as a web user eager for automatic services, this agent can prefer to propose to such an user to query the catalogs ($\text{Cap}_2$) instead of browsing them ($\text{Cap}_1$). For example, the class of user, that have clear in mind the product they are looking for, may prefer to build up a query specifying some product characteristics.

**fast search on catalogs.** This is the case a Customer asks for a specific product and wants to get it quickly; hence, $\text{Cap}_2$ may help better than $\text{Cap}_1$ to achieve such non-functional requirement, i.e. a QoS.

**simple search on catalogs.** Such a softgoal is suitable to model a system strategy in order to deal with an adaptive behavior. For example, if a failure occurs when the Customer queries the catalogs (i.e. $\text{Cap}_2$), the system may react with the strategy of proposing a simple search on such catalogs (i.e. $\text{Cap}_1$); namely, avoiding the user to specify necessary product info (may be the failure reason).

According to the design explained above, the software agent **Search Manager** has to be aware of the fact: it cannot contemporary make true the two softgoals search for desired product automatically and simple search on catalogs. That is, the contribution links technique allows to identify conflicting goal conditions that we aim to reflect into the agent knowledge-base implementation.

The **Tropos** concepts, describing the opportunity part of the capability, naturally accommodate with BDI agent concepts as partially illustrated in Fig. 2. For example, Fig. 2 shows an ADF fragment for the **Search manager** capabilities definition in terms of couple $(\text{goal,plan})$. To be more precise, such an ADF fragment focuses more on the opportunity part of the capability, specifying the plan to be executed each time a goal is triggered. To deliver on such a task, the ADF also specify a plan responsible for the choice of the right ability associated to the selected goal, e.g. in the case of $\text{provide product info}$ there are two alternatives (abilities) browse catalog (belong to $\text{Cap}_1$) and solve query (belong to $\text{Cap}_2$). As already said, in this approach a criteria to deal with alternatives selection is given by the use of softgoal contributions. Therefore, we have reflected the information of capability design in the agent knowledge base in terms of its beliefs. As you can see from Fig. 2, the ADF beliefs section embeds all such design information as a capability-table (a list of all capabilities) stored into the belief-base. Operatively, each time a goal of a capability is triggered, the agent may read from its belief-base what plan (ability) gives a stronger contribution to the softgoal achievement. In our experiments, the scenario illustrated in Fig. 1, for $\text{Cap}_1$ and $\text{Cap}_2$ along with softgoal contribution relationships (bold shapes), has been generated, as described in the next Section.

### 3.2 Implementation

Basic features of the MAS implementation, generated with our design framework, are:

- **a)** Each agent can play different roles according to special request-messages, related to target goals. That is, an agent can sense the environment and consequently switch to a specific role, execute the most appropriate capability, chosen by using the knowledge coded into the opportunity part, i.e. beliefs base. Notice that, this behavior (at run-time) can be related to the opportunity analysis, as specified at design time.

- **b)** The ability part of a capability is implemented as a specialization of the class **jude.core.behaviours.FSMBehaviour**, namely it represents a finite states machine (automaton). This allows to exploit monitoring mechanisms, during capability execution, in order to make the agent aware of failures and able to react to them. In this case a mapping between design

---

**Figure 2. ADF fragment of belief base and capability definition.**
artifacts (Tropos plan /AUML activity diagrams) and run time artifacts (finite states machine) has been defined.

Turning to our example, each time an event triggers the goal provide product info, the agent has to sense environmental conditions (in terms of active softgoals) to select among (abilities of) Cap1 and Cap2. In order to test the opportunity part of the capability Cap2, we randomly activated the designed softgoals (stored as agent beliefs): search for desired product automatically, fast search on catalogs, and simple search on catalogs. Then we can observe the agent when choosing the right plan (ability) that maximally contributes to the achievement of the active softgoal, according to its beliefs-base. Hence, cumulative values of contribution relationships related to each capability (shown in Fig. 1) are computed.

4. Related Work

Among the research lines of interest for the work presented in the paper, we will focus on agent-oriented approaches for complex system development and on the traceability of stakeholder needs since the requirements phases. In the last years, agent-oriented software engineering (AOSE) methodologies have adopted different high level design concepts to represent system requirements and to support different design phases [3]. For example, in MaSE (Chapter 11 in [3]), authors do not focus on requirements analysis, on the contrary, they are principally interested in dealing with the system architecture in terms of both agent roles and agents coordination level. On the contrary, Gaia [10], supports the requirements analysis, but it lacks in showing how these requirements are correlated to design choices about agents’ capabilities. Thus, it seems that agent-based development methodologies have weakly faced the issue of describing how stakeholders’ needs may drive the definition and the design of agent capabilities. In the Passi methodology (Chapter 4 in [3]) the process that guides the agent-based code generation is quite similar to our approach. But, while Passi does not considers stakeholder intentionalities and social dependencies as strategic knowledge elements to effectively deal with the capability selection, we are able, by modelling the opportunity concept, to embed in the agent knowledge also stakeholder needs and environmental constraints figured out at the early phases of the requirements analysis.

5. Conclusions and Future Work

This paper describes a tool-supported methodology for developing MAS systems able to adapt to stakeholder needs and domain constraint changes. We revisited the Tropos agent capability definition in order to correlate it with stakeholder needs (opportunity). Moreover, exploiting the new capability design, Tropos allows the designer to trace stakeholder needs and domain constraints from the requirements analysis till the implementation phase. In order to make the agent aware about stakeholder needs (modelled as softgoals) and their influences on capability selection, we represent them in the knowledge base of a BDI agent (as its beliefs). Specifically, our approach has been illustrated via an on-line selling scenario that focuses on the opportunity part of the capability design and implementation.

As future work, we are coping not only with agent capability generation, but also with an automatic implementation of a complete deliberative agent, reflecting such advances in our agent development tool: TAOM4E [5, 6].

References