

Refining Goal Models by Evaluating System Behaviour

Mirko Morandini, Loris Penserini, Anna Perini, and Angelo Susi

Fondazione Bruno Kessler - IRST, Via Sommarive 18, I-38050, Trento, Italy
{morandini,penserini,perini,susi}@itc.it

Abstract. Nowadays, information systems have to perform in complex, heterogeneous environments, considering a variety of system users with different needs and preferences. Software engineering methodologies need to cope with the complexity of requirements specification in such scenarios, where new requirements may emerge also at run-time and the system's goals are expected to evolve to meet new stakeholder needs.

Following an agent-oriented approach, we are studying methods and techniques to design adaptive and evolvable information systems able to fulfill stakeholders' objectives.

In a previous work we defined an Agent-Oriented framework to design and code system specifications in terms of goal models and we instantiated it in a tool supported process which exploits the Agent-Oriented Software Engineering methodology *Tropos* and the Multi-Agent Platform JADE/Jadex [11].

In this paper, we show how to use this framework to develop a system following an iterative process, where the system execution allows enriching the system specification given in terms of goal models.

Experimental evaluation has been performed on a simple example and lead to the refinement of the designed goal model upon the analysis of the system's run-time behaviour.

1 Introduction

Information systems are today expected to perform in complex environments which make computing resources available to anyone, at any time and anywhere. In these scenarios, complexity comes from the variety of system users (including organizations) with their needs and preferences, which tend to evolve according to the dynamic nature of users in the network, and from the heterogeneity of the environment a system is deployed in. Therefore, systems should be aware of users' goals and able to choose the most suitable behaviour from various alternatives.

These scenarios motivate research on practices and methodologies for software development. Traditional software development models, which assume that the requirements specification has been finalized before proceeding to design and then to implementation, need to be replaced by more flexible iterative models, able to take into account that new requirements may emerge also at run-time. Moreover, the traditional concept of software maintenance has to be revised since systems are expected to evolve to meet the needs of the changing environment rather than to preserve their original structure [8].

Multi-Agent Systems (MAS) provide candidate technologies for building software with adaptivity and evolvability qualities [6], while recently proposed Agent-Oriented Software Engineering (AOSE) methodologies offer a complementary paradigm for the analysis of system requirements and design [4]. Some of them, such as GAIA [17] and *Tropos* [1] offer concepts and models to analyse the system and its environment in terms of agent organizations. Moreover, *Tropos* has been recently proposed as a methodology to support “high-variability software design” through the explicit modelling of the different alternative design solutions to a given stakeholder goal (requirement) [7].

We are studying methods and techniques to design information systems with qualities such as adaptivity and evolvability, following an Agent-Oriented approach. That is, we conceive an information system as an open network of software agents who interact with each other and with human/organizational agents in their operational environment in order to fulfil stakeholder objectives. Concretely, we propose a development framework which adopts MAS technology as implementation platform and agent-oriented methods and techniques for the analysis and the specification of system requirements and design. In [11, 12] we instantiate this framework with respect to the *Tropos* methodology and to the Jadex/JADE platform [14], and propose a tool-supported process to derive agents that base on a Belief-Desire-Intention architecture [15] from *Tropos* goal models.

In this paper, we show how this framework can be used to develop a system following an iterative process, in which the system execution allows to enrich the system specification expressed in terms of goal models. From a goal-oriented system model we derive agent skeletons automatically in a tool-based process. We execute the modelled system, simulating system users and observing system behaviour in correspondence to variability in user desires and in environmental conditions. The different MAS behaviours can then be traced back to the specification of the alternatives in the goal model, giving experimental evidence of the effectiveness of the proposed framework in supporting traceability between run-time and design-time artefacts. Moreover, run-time observations could lead to a refinement of the design. For instance contribution relationships between model elements can be further qualified or quantitative analysis of system qualities, which have been defined at design time in terms of system (soft)goals, can be performed. We consider this work as a first step towards setting up feedback mechanisms from run-time to the design, a core aspect in the development of adaptive systems.

The paper is structured as follows. In Section 2, we recall basic concepts of the proposed development framework and describe the tool-supported process, which exploits the *Tropos* methodology and the JADE/Jadex MAS platform. A simple travel agency system is used as example to illustrate the approach. The system handles requests from different customer categories and gives proposals for a full travel package, according to user preferences. Experimental evaluations, based on a run-time simulation, are described and discussed in Section 3. Related work is presented in Section 4 and concluding remarks in Section 5.

2 Background

2.1 Conceptual Framework

We adopt concepts from recently proposed AOSE methodologies [4] and from BDI MAS research [6] to define an Agent-Oriented approach to system design and coding. The Belief-Desire-Intention (BDI) architecture, as proposed in [15], bases on three mental concepts: *beliefs* which model the knowledge of an agent about himself and about its environment, *goals* the agent can try to achieve, and *intentions*, sets of plans an agent commits for execution to achieve a goal. Within our framework, we further use knowledge level concepts such as those of *agent*, which can be social, organizational, human or software and *social dependency* that defines the obligations of an agent to others.

The key part in the analysis and design stages is the so called *Goal Model (GM)*. Like in other approaches [5, 16], in our framework a *GM* is a goal graph consisting of a forest of AND/OR decomposed goals, 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. During requirements analysis, *GMs* make easier to model stakeholders' goals and their relationships, showing how they really affect the system functionalities. Moreover, deriving an agent *GM* at architectural design allows designers to model dependencies between the system agent goals and stakeholder goals.

More operative concepts are needed during software agents specification. We define a *capability* as the sub-graph rooted in a leaf goal containing a 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. Therefore, a *GM* can be also seen as a schema for the possible behaviours an agent can use to fulfil its goals.

More formally, let E be the set of events¹ 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, Cp a set of capabilities an agent can exploit, and $G_L \subseteq G$ the set of leaf-level goals an agent can operationalize, we give the following definitions:

behaviour-schema *BS* is the set of all possible type of behaviours an agent can play $BS = \{Bh_1(X), \dots, Bh_n(X)\}$, where each Bh_i is a sub-schema representing a set of behaviours associated to a specific set of events and constraints. Formally, X is a list of attributes $X = \{Events, Constraints, Goals\}$, where *Events* assumes values in 2^E ; *Constraints* assumes values in 2^C , and *Goals* assumes values in 2^{G_L} .

behaviour-schema function A behaviour-schema function f_{Bh} associates a set of events and constraints to a set of leaf-level goals: $f_{Bh} : 2^E \times 2^C \rightarrow 2^{G_L}$. It allows to build the $Bh(X)$ specific to an occurring event and the constraints perceived.

¹ Examples are goal triggering messages.

capability function For each sub-schema Bh , there exists a capability function $f_{Cp} : Bh \rightarrow 2^{Cp}$ such that, giving in input a behaviour b_i , retrieves the different sets of Capabilities an agent can execute to exhibit this behaviour.

Behaviour-schema- and capability-functions allow to query the behaviour-schema and a GM structure about agent properties with reference to concrete instances of behaviours and capabilities, each time an event occurs or environmental conditions change. For example, each time an agent receives a request message (an event), it interprets it in order to extract the goals to be triggered, and concurrently perceives environmental conditions (C) to better choose the right behaviour.

2.2 Tool Supported Framework

In [11, 12] we instantiate the described framework using the *Tropos* methodology for analysis and design, and JADE and the Jadex BDI agent platform for the implementation. The *Tropos* agent-oriented methodology borrows modelling and analysis techniques from goal-oriented requirements engineering approaches and integrates them into an agent-oriented paradigm (see [1] for details). The main idea in *Tropos* is to support knowledge level specification by providing a conceptual modelling language which offers concepts like *actor*, *goal*, *plan*, *resource*, *capability*, and *social dependency* between actors. The methodology provides a graphical notation to depict views of a model, along with analysis techniques and supporting tools [13].

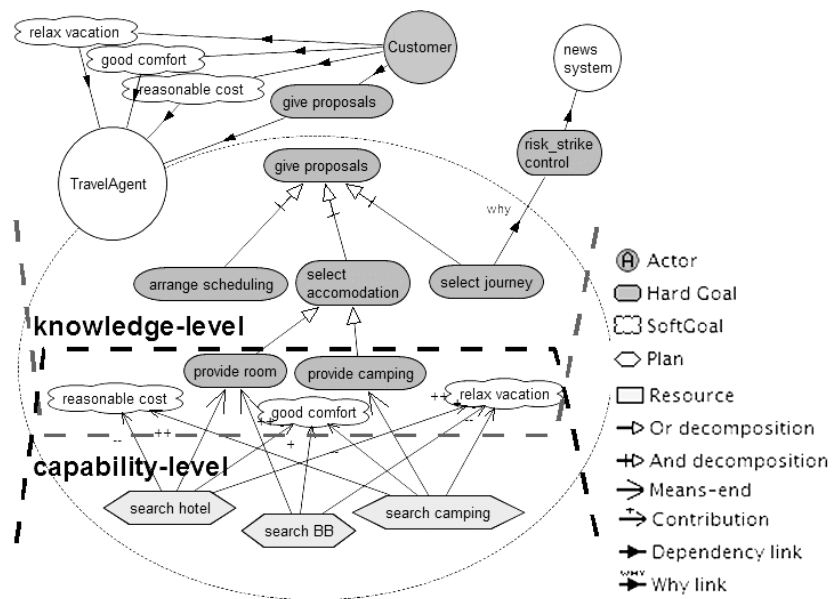


Fig. 1. Tropos architectural design: a fragment of knowledge and capability levels from the goal model of agent TravelAgent.

In the rest of this section, we give an overview of the framework by defining basic process artefacts and their role in the main process stages, as depicted in Figure 2. Moreover, we give more details on how a *GM* is automatically mapped into a BDI agent specification that can, at execution time, give useful feedback to refine the original design. To illustrate our approach we take as an example a simple travel agency system, *TravelAgent*. The system handles requests from several classes/categories of customers (e.g. business customers, vacation customers or students) and gives proposals for a full travel package, according to the users' preferences. These preferences are modelled through softgoals. For example, as illustrated in Figure 1, possible softgoals to characterize a customer category are reasonable cost, good comfort, and relax vacation. The agent tries to achieve them exploiting different alternatives for journey and accommodation and selecting suitable additional activities.

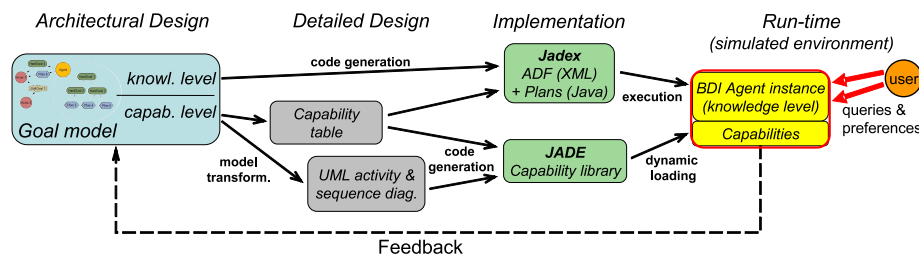


Fig. 2. Process's artefacts and their role in *Tropos*. Feedback from system execution to design can be obtained upon observing the system behaviour.

Process Artefacts Adopting *Tropos* in our framework allows us to represent MAS with *GMs*, resulting from the analysis of each actor's point of view. *Tropos GMs* are based on the *i** notation [16]. They are represented in terms of a forest of AND- and OR-decomposed goals, along with lateral contributions labelled +, - (for partial positive or negative contribution to the satisfaction of a goal) or ++ (for strong positive or negative contribution). For example, if $g_1 \xrightarrow{++} g_2$ and g_1 is fulfilled, so is g_2 . Additionally a *GM* contains means-ends relationships among plans and goals, to define the means to satisfy a goal.

In *Tropos*, *GMs* are built during requirements analysis to characterize domain stakeholders, their needs and their dependencies from the system-to-be. In the architectural design phase they are used to detail software agents. As this paper work focuses on getting feedback to design-time from the generated agents, here we consider mainly *GMs* at the *Tropos* architectural design phase. An example is given in Figure 1, which illustrates a fragment of a *GM* for the *TravelAgent*, which represents the main actor of the software system under development.

At *Tropos* architectural design phase, a *GM* represents the agent intentionality in terms of how the agent perceives the environment, applies strategies to fulfil its responsibilities, and chooses alternative ways to adapt to requirements changes. In other

words, like a human being, the agent perceives the environment and chooses a suitable behaviour. Notice that the set of possible agent behaviours can be characterized in terms of perceivable events, environmental conditions and agent goals by applying the behaviour-schema function, f_{Bh} , to the GM . The actual behaviour results from the execution of the specific capabilities thanks to the capability function, f_{Cp} .

Figure 1 depicts the two different abstraction levels that characterize the agent design: *knowledge level* and *capability level*. The knowledge level refers to the goal AND/OR decomposition part of the GM that contributes to the description of the behaviours the specific agent role can play. The capability level brings about the executable part of an agent and its connection to the agent's leaf goals.

Capabilities ($cp \in Cp$, where Cp is the set of all capabilities of the agent) represent the glue between the two agent modelling levels. A capability is defined by the concepts of *ability* and *opportunity*. The *ability* refers to the plans for achieving a given goal and is specified by a means-end relationship between the goal and the plan. The *opportunity* represents user preferences (contributions between goals and plans to softgoal, $c \in 2^C$) and environmental conditions represented by message events $e \in 2^E$ that affect the agent's beliefs. At run time, these preferences and conditions can enable or disable the execution of an ability.

Following the design process sketched in Figure 2, a *capability table* extracted from the GM , and UML 2.0 diagrams, extracted by model transformation from the GM s capability level, are used to generate JADE code in a capability library. The structure of an agent's reasoning part, that relies on a BDI architecture, can be automatically generated from the knowledge level of the GM .

Capabilities	Means_End(goal,plan)	List of Contributions
cp_1	provide room, search hotel	{reasonable cost --;good comfort ++; relax vacation --}
cp_2	provide room, search BB	{good comfort ++;relax vacation +}
cp_3	provide camping, search camping	{reasonable cost ++;good comfort --; relax vacation --}
...	...	{...}

Table 1. Fragment of TravelAgent capability Table.

Table 1 depicts a fragment of the capability table for the TravelAgent example, which was obtained from the GM in Figure 1. If the goal *select accommodation* is triggered, as this is OR-decomposed in the two sub-goals, the agent TravelAgent has two possible behaviours to satisfy the triggered goal: one that can achieve the goal *provide room* (by cp_1 or cp_2) and one that can achieve the goal *provide camping* (by cp_3).

The design artefacts (GM s, capability table, activity- and sequence diagrams) drive agent code generation. Specifically, our tool supported framework allow us to generate a *library of capabilities* from the capability table and the activity- and sequence diagrams, as detailed in [12]. Executable skeletons for the *BDI agents*, which are able to use the capabilities in this library, can be automatically generated from the knowledge

level specification contained in the *GM*, through a mapping of the *GM* structures into a BDI agent description for the *Jadex* framework, specified by an Agent Description File (ADF) in XML format, augmented with some Java code².

The implementation consists of *Jadex* BDI agent definitions along with their capabilities. For the simulation we run instances of these agent definitions. The agents are queried by simulated users sending request-messages and the resulting behaviour is then used to give feedback to the design artefacts.

Coding the GM into BDI Agents In order to endow the generated BDI agents with all the information included in the *GM*, 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). A sketch of the mapping is given below, while we remind the interested reader to see [11] for more details.

Goal. As a *Jadex*-goal can be only triggered by a *Jadex*-plan, a *Tropos* goal is mapped to a pair of $\langle goal, plan \rangle$ in *Jadex*.

Softgoal. Softgoals are considered as abstract entities related more to beliefs and desires than to goals and plans. 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 model domain constraints $c \in C$ to drive the selection of the most convenient behaviour $b \in Bh(X)$, once an event $e \in 2^E$ 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. This value expresses the softgoal's actual importance and may change from time to time to reflect environmental changes.

Plan. *Tropos* plans that have a direct means-end relationship to leaf goals (root-level plans) are mapped to *Jadex* plans according to our definition of capability.

AND-decomposition. If an AND-decomposed goal is activated, all subgoals have to be dispatched. The following *Jadex* solution was adopted: an AND-decomposed goal is set as trigger for exactly one plan, called AND-dispatch-plan (Figure 3). In the plan body, all subgoals have to be dispatched in (random or user defined) 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 techniques for compensation of already executed actions have been considered.

An analogous mapping for the **OR-decomposition** is described in [11], while *Tropos* **means-end** relationships are 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 root-level plans in *Tropos*, namely those required to build up agent capabilities. *Jadex* supposes that every applicable plan for a goal is able to satisfy that goal completely. Therefore, if more than one plan is applicable, *Jadex* meta-level reasoning is exploited to select the appropriate plan, as in the case of alternative paths in OR-decomposition. The selection

² Further details can be found in [14].

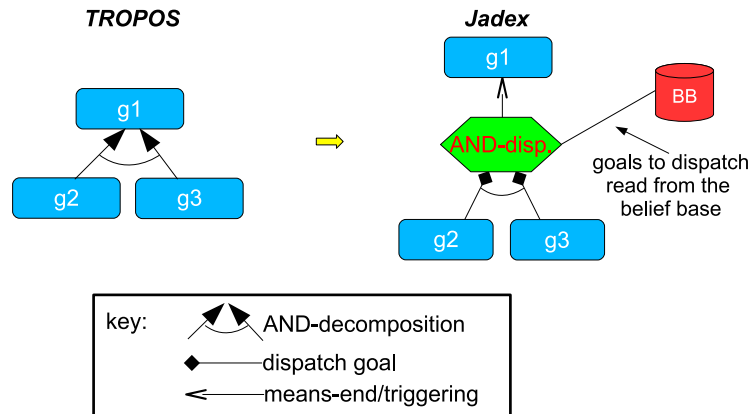


Fig. 3. Mapping of the *Tropos* goal AND-decomposition into an equivalent Jadex BDI structure.

of alternatives can be guided by preconditions and by softgoal contributions.

The generated agent can evaluate costs for every goal and plan. They include softgoal contribution and importance: negative contributions cause higher cost, lower importance of the relative softgoal to the user can alleviate this penalty. Moreover, each generated agent endows knowledge about its goal relationships from the *GM* in its belief base: AND/OR decompositions, dependencies, delegations, and contribution links.

3 Experimental Setting and Evaluation

This Section describes a general experimental setting, suitable to be applied in several kinds of scenarios, and the results obtained.

The objective of our experiment is twofold: on one hand, we aim at verifying the behaviour of a MAS with respect to the designed specifications; on the other hand, we aim at supporting the refinement (such as the introduction of new relationships) of the *GM* by exploiting information retrieved from the simulation.

We refer to the *TravelAgent* example, partially depicted in Figure 1. The main idea is to focus on the preferences of different customer categories recognized by the system at run-time by profiling users from the set of queries they submitted. We observe the system while it is adapting to each category, trying to maximize customer satisfaction (customer's softgoals delegated to the *TravelAgent* system) and providing evidence of how such softgoals have different impact to the system's own internal softgoals (e.g. maximize profit).

3.1 Experimental Setting

We refer to the fragment of *GM* shown in Figure 1. Suppose that a generic *Customer* could be distinguished into three categories: *business customer* (BC), *vacation customer*

(VC) and *student customer* (SC), each one composed by individuals having similar preferences and similar requests to the travel agency system. The simulation assumes that these categories will be recognized by the system only at run-time.

In the simulation, the **Customer** (in the following also *user*) interacts with the system by submitting sets of queries that correspond to set of activation events for system goals (belonging to 2^E). Moreover, the system is supposed to acquire information from the environment in order to provide user profiling. This would allow to activate sets of softgoals that represent the users' preferences (belonging to 2^C). Basing on this information our system is able to assume the best-suited behaviour, activating all necessary capabilities. Table 2 shows the components for the run-time choice process of sets of capabilities. In particular, the second column represents the input elements. The former is given as a set of queries made by the different Customer Categories (i.e. $CC = \{BC, VC, SC\}$), e.g. the business customers' query will be q^{BC} , whereas the latter is given by the set of user preferences and constraints (e.g. C^{BC}) perceived by the system via an user profiling activity or by user-guided configuration. The third column contains the set of behaviour instances suitable for each category (e.g. b_i^{BC}) related to the queries (e.g. q_i^{BC}), while the fourth column contains the capability sets (e.g. Cp_i^{BC}) able to realize the corresponding behaviours.

To choose appropriate capabilities, the system exploits the *behaviour-schema* function f_{Bh} and the *capability* function f_{Cp} , defined in Section 2. In particular, the first line of the table refers to a single query made by an user that belongs to the BC category. Let us assume that such a category is characterized by a set Q^{BC} of m possible queries ($Q^{BC} = \{q_1^{BC}, \dots, q_m^{BC}\}$) and by a set C^{BC} of preferences ($C^{BC} = \{sg_1^{BC}, sg_2^{BC}, \dots\}$), which are inputs for the application of the *behaviour-schema* function. With this input, the system is able to compute the set of possible behaviours b_{1-m}^{BC} , that can be exploited in order to accomplish user requests.

The application of the *capability function* f_{Cp} to a behaviour b_i allows the system to retrieve the different sets of capabilities that can be activated in order to operatively execute this behaviour. The capability to execute can then be selected according to the opportunities. This calculation process can be repeated for all user categories in the model.

User class (CC)	Trigger events ($2^E \times 2^C$)	Behaviours ($Bh(X)$)	Capability sets (2^{Cp})
BC	q_1^{BC}, C^{BC} ...	b_1^{BC} ...	Cp_1^{BC} ...
	q_m^{BC}, C^{BC}	b_m^{BC}	Cp_m^{BC}
VC	q_1^{VC}, C^{VC} ...	b_1^{VC} ...	Cp_1^{VC} ...
	q_n^{VC}, C^{VC}	b_n^{VC}	Cp_n^{VC}
SC	q_1^{SC}, C^{SC} ...	b_1^{SC} ...	Cp_1^{SC} ...
	q_k^{SC}, C^{SC}	b_k^{SC}	Cp_k^{SC}

Table 2. System inputs (CC and $2^E \times 2^C$) and outputs ($Bh(X)$ and 2^{Cp}) for the simulation, following the definition recalled in Section 2.1

A domain expert would be able to define contribution relationships between each capability and internal softgoals, such as maximize profit, but she cannot know *a-priori* which capabilities will be executed to satisfy a user query that occurs to the system.

C_i	Vacation Customer	Student Customer	Business Customer
good_business_travel	0	0	1
good_comfort	0.6	0.2	1
action_vacation	0.4	1	0
good_time_utilization	0.3	0	1
reasonable_cost	0.6	1	0.1
relax_vacation	0.6	0.3	0

Table 3. Softgoals used by the system to profile the user preferences. These values are supposed to be given by domain experts.

In Table 3 sample values for the importance of the contributions to softgoals adopted to characterize the customer categories are given; those values refer to the three different classes of customers and are expressed via numeric values.

We prepared the experiments, defining a set of queries for every different customer category and identifying sets of softgoals that are typical for them. Table 4 gives examples for sets of queries for the three classes of users we considered. Moreover, for each capability (cp), we define its contribution towards the internal system softgoals ($sgint_1, \dots, sgint_l$), as illustrated in Table 5.

Query	Vacation Customer	Student Customer	Business Customer
q_1	give proposals	give proposals	give proposals
q_2	provide camping, car journey, prop. act.	propose activities	provide room, flight journey
q_3	provide room, train journey	provide room, flight journey	select journey
q_4	select accommodation	camping, train journey	select accommodation

Table 4. Queries that characterize each customer category.

Capability	Contribution to maximize profit	Contribution to maximize travel miles
search hotel	0.8	0
search BB	0.4	0
camping	0.3	0
eurostar train	0.2	0.3
intercity train	0.1	0.2
business flight	0.9	1
low cost flight	0.2	0
gastronomy	0	0
nightlife	0	0
...

Table 5. Contribution values between each capability and the internal softgoals.

Table 6 can be built only after running the system along with simulated inputs and getting information on which set of capabilities were executed by the system to satisfy user queries. Specifically, the table shows the schema of the relationships among sets of capabilities and internal softgoals ($sgint_1, \dots, sgint_l$), for each agent. These values are then used to compute the cumulative contribution of the run-time sets of capabilities cp , belonging to the selected behaviour b_i , to a given internal softgoal, via the functions $f_{avg}(val_{1,j} \dots val_{m,j})$ (at the bottom of the table). In our experimental setting we used the function:

$$avg(val_{1,j} \dots val_{m,j}) = \sum_{i=1}^m (val_{i,j})/m \quad (1)$$

Query	Sets of Capabilities	sgint ₁	...	sgint _l
q_1^{BC}	Cp_1^{BC}	$val_{1,1}$...	$val_{1,l}$
...
q_m^{BC}	Cp_m^{BC}	$val_{m,1}$...	$val_{m,l}$
		$f_{avg}(val_{1,1} \dots val_{m,1})$...	$f_{avg}(val_{1,l} \dots val_{m,l})$

Table 6. Contribution relationships among capability groups and internal system softgoals.

Notice that an analysis by simulation does not cope with the possible contribution produced by all the different capability groupings. On the contrary, the simulation will converge towards the only sets of capabilities requested by the real customer categories.

3.2 Results and Discussion

After the simulation, the set of data related to the experiment for the *TravelAgent* scenario has been collected. According to our first objective, we are able to monitor the system behaviour (b), each time a query (e.g. q_4^{BC}) occurs, along with some user preferences (e.g. $C^{BC} = \{\text{good comfort}\}$), verifying that b belongs to the *GM* behaviour schema ($b \in Bh(X)$). Specifically, we can observe that the system has the ability to adapt its behaviour to best accommodate with the current customer category. For example, let us assume that q_4^{BC} will trigger the system goal *select accommodation*, along with the softgoal *good comfort*. Now, the system is able to navigate the *GM* in order to maximize the user preference modelled by this softgoal.

Looking at the *GM* fragment illustrated in Figure 1, we can see that the goal *select accommodation* has two alternative ways to be achieved, i.e. *provide room* and *provide camping*. The system will first try to select *provide room*, because its capabilities (characterized by the two plans *search hotel* and *search BB*) give the biggest contribution to the given user preference. The same procedure will be used in a next step to discriminate between the two available capabilities, this time resulting in the selection of *search hotel*.

These experiments confirmed the effectiveness of the framework in supporting traceability between run-time and design-time artefacts.

To meet our second objective, we simulate the execution of a set of user queries and preferences in order to revise softgoal relationships in the *GM*. Table 7 shows the

sets of capabilities activated by the system, i.e. the behaviour instances it selected at run-time, as a response to the simulated user queries described in Table 4. In Table 7, each row specifies a query from a particular category of users (BC, VC and SC). Contributions to *maximize profit* are calculated by summing the value of each capability contribution as indicated in Table 5, e.g. in the case of Cp_1^{BC} : *eurostar train + search hotel + gastronomy* = 0.2 + 0.8 + 0 = 1.

Query	Sets of executed capabilities	Contribution to <i>maximize profit</i>
q_1^{BC}	Cp_1^{BC} : eurostar train, search hotel, gastronomy	1
q_2^{BC}	Cp_2^{BC} : business flight, search hotel	1.7
q_3^{BC}	Cp_3^{BC} : eurostar train	0.2
q_4^{BC}	Cp_4^{BC} : search hotel	0.8
		$f_{avg} = 0.925$
q_1^{VC}	Cp_1^{VC} : low cost flight, search BB, culture	0.6
q_2^{VC}	Cp_2^{VC} : use own car, camping	0.3
q_3^{VC}	Cp_3^{VC} : intercity train, search BB	0.5
q_4^{VC}	Cp_4^{VC} : search BB	0.4
		$f_{avg} = 0.45$
q_1^{SC}	Cp_1^{SC} : low cost flight, search camping, nightlife	0.5
q_2^{SC}	Cp_2^{SC} : nightlife	0
q_3^{SC}	Cp_3^{SC} : low cost flight, search BB	0.6
q_4^{SC}	Cp_4^{SC} : intercity train, camping	0.4
		$f_{avg} = 0.375$

Table 7. Capability groups associated to every query at run-time.

The results of these queries allow to observe how the real (in our case simulated) customer preferences affect system behaviour.

The capability groups corresponding to the different behaviours of the TravelAgent can then be used to add or quantify *Tropos* contribution links.

Figure 4 A), shows the values of f_{avg} computed according to 1, as shown in Table 7, considering the internal softgoal *maximize profit*. In Figure 4 B), a softgoal *customer satisfaction* was introduced to aggregate the softgoals relevant to a specific customer category. Contributions between them and the internal softgoal *maximize profit* can be drawn and quantified by the contribution values computed at run-time.

This result can contribute both to validate existing contribution links and to add new ones. In the case run-time feedback is in contrast with the design-time models, a revision of the *GM* could be required.

In a subsequent step, these new relations could be used by the system to adapt its strategic behaviours, not only according to the user preferences (i.e. softgoal *customer satisfaction*), but also according to its internal organizational objectives (i.e. softgoal *maximize profit*), following a trade-off for the achievement of this two softgoals.

4 Related work

Different research lines are of interest to the work described in this paper. Here we focus on research in AOSE methodologies, which aims at supporting traceability between process artefacts, and research on methods for evaluating design strategies.

Along the first research line, we shall mention the Prometheus methodology [9], which makes use of goal models to describe system requirements. Analogous to [1],

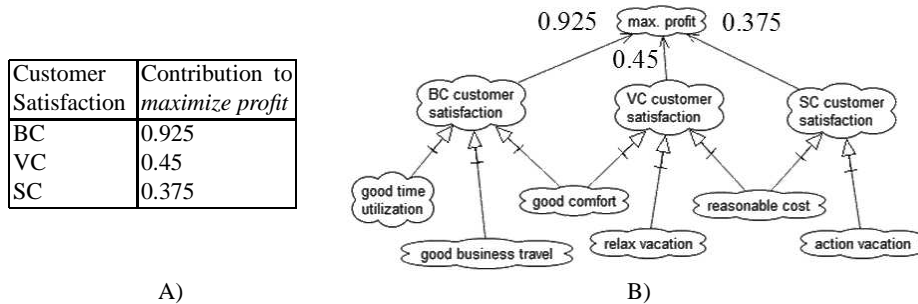


Fig. 4. A) Quantifying the contribution relationships between each customer satisfaction softgoal and the maximize profit softgoal; B) visualizing the results in terms of the *Tropos* goal model refinement. The labels define the new relationship values.

after building a goal model, the designer identifies those goals that are related to system functionalities (by the use of *descriptors*) and delegates them to specific system actors. Then functionalities are grouped to characterize scenarios, namely sequence of steps (functionalities) in order to achieve a goal. Notice that, this grouping mechanism is also used to determine different agent types (roles). Agents' awareness about their goal model is limited in Prometheus. For example, designed agent behaviour 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 behaviours. Moreover, also traceability from and to design artefacts is not supported.

Hermes [2] aims at overcoming the weak points of the above-mentioned methodology, considering the goal model a core element of the implemented agent. In Hermes, generated BDI agents are aware of their goal model, called *Interaction Goal Hierarchy Diagram*, which is used to characterize behavioural strategies to cope with social commitments. This gives a more flexible approach in respect to traditional message-centric agent interaction approaches. Hermes needs further research in order to deal with a complete design framework, actually it does not cover the requirements analysis and architectural design phases.

Along the second research line, several approaches have been proposed to evaluate the different design strategies used in a *GM*, e.g. to achieve goals [3, 10]. In [10] the authors propose three different evaluation criteria (*symbolic*, *scenario based*, and *quantitative*) to characterize the cooperation strategies of an agent-based P2P system. While the *symbolic* and *scenario based* criteria are fulfilled at design time, the *quantitative* criterion takes advantage from run-time results. In particular, the *symbolic* evaluation criterion is elaborated through analysis of the contribution links in the agent *GM*. A substantial difference to our approach is that their evaluation system has not been automatically generated from design-time artefacts (e.g. *GM*). Besides, they have not discussed how to correlate the run-time feedback to design-time *GM*.

The approach presented in [3] proposes a formal framework to reason on generic *GMs*, namely not only on those related to software systems. Specifically, the authors adopt some well-known algorithms to navigate the *GM* relationships, proposing an as-

assessment criterion to propagate contribution values (labels) in order to verify the goal achievement. The analysis is called *qualitative* if the labels range in $\{++, +, -, --\}$, while it is called *quantitative* if the labels assume numeric values. The most significant difference to our framework is that their analysis framework works only when applied to a *GM* at design-time, while our approach considers run-time as a principal source of feedback to the design.

5 Conclusions and future work

In this paper we described an Agent-Oriented framework for developing systems with qualities such as adaptivity and evolvability. We showed how information gained from the execution of a system can be used to refine the original design.

As example, we modelled a simple travel agency system. The system handles requests from different customer categories and gives proposals for a full travel package, according to user preferences. We tested our approach by simulating an environment where several categories of users, which are characterized by their own preferences and typical requests, interact with the MAS. We observed the behaviour of the system in order to verify that it is compliant with the designed specifications. This confirmed the effectiveness of the framework in supporting traceability of *Tropos* concepts between run-time and design-time artefacts. Moreover, starting from the run-time behaviour of the system in response to the user queries, we described a way to refine and extend the relationships among a set of user preferences and the internal goals of the system.

We believe that this is a first step towards defining feedback mechanisms from the real execution of the system back to design.

As future work we will revise the proposed framework, formalizing it and investigating on some AI technique which will allow the system agent to automatically discriminate the customer category (i.e. by user profiling) from a set of input queries. Moreover we aim at experimenting the framework in a real environmental setting.

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. C. Cheong and M. Winikoff. Hermes: Designing Goal-Oriented Agent Interactions. In *In the proceedings of the 6th International Workshop on Agent-Oriented Software Engineering (AOSE-2005) Utrecht, colocated with AAMAS05*, 2005.
3. P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Reasoning with Goal Models. In *in Proc. of the 21th International Conference on Conceptual Modeling (ER)*. Springer, October, 2002.
4. B. Henderson-Sellers and P. Giorgini, editors. *Agent-Oriented Methodologies*. Idea Group Inc., 2005.
5. N. Jennings. *Foundations of Distributed Artificial Intelligence*, chapter Coordination Techniques for Distributed Artificial Intelligence. Wiley-IEEE, 1996.
6. N. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

7. A. Lapouchnian, S. Liaskos, J. Mylopoulos, and Y. Yu. Towards Requirements-Driven Autonomic Systems Design. In *Design and Evolution of Autonomic Application Software (DEAS'05) at ICSE 2005*, 2005.
8. P. Norvig and D. Cohn. Adaptive software. *PC AI*, 11(1):27–30, 1997.
9. L. Padgham and M. Winikoff. Prometheus: A practical agent-oriented methodology. In B. Henderson-Sellers and P. Giorgini, editors, *Agent-Oriented Methodologies*. Idea Group, 2005.
10. L. Penserini, L. Liu, J. Mylopoulos, M. Panti, and L. Spalazzi. Modeling and Evaluating Cooperation Strategies in P2P Agent Systems. In *in Proc. of the International Workshop on Agent and Peer-to-Peer Computing (AP2PC 2002)*. Springer, LNCS 2530, July, 2002.
11. L. Penserini, A. Perini, A. Susi, M. Morandini, and J. Mylopoulos. A Design Framework for Generating BDI-agents from Goal Models. In O. Sheory and M. Huhns, editors, *6th International Joint Conference on Autonomous Agents and Multi-Agent Systems, AAMAS'07, Honolulu, Hawai'i*, 2007. Extended version available as ITC-irst TR200601002 at <http://sra.itc.it/images/sepapers/bdiagents.goalmodels.pdf>.
12. 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 (CAiSE'06)*, volume 4001 of LNCS, pages 465–479, Luxemburg, 2006. Springer-Verlag.
13. A. Perini and A. Susi. Automating Model Transformations in Agent-Oriented Modelling. In *Agent Oriented Software Engineering VI: AOSE'05*, volume 3950 of LNCS, pages 167–178. Springer-Verlag, 2006.
14. A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: A bdi reasoning engine. In J. D. R. Bordini, M. Dastani and A. E. F. Seghrouchni, editors, *Multi-Agent Programming*, pages 149–174. Springer Science+Business Media Inc., USA, 9 2005. Book chapter.
15. A. S. Rao and M. P. Georgeff. Modeling rational agents within a bdi-architecture. In *KR*, pages 473–484, 1991.
16. E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, Department of Computer Science, University of Toronto, 1995.
17. F. Zambonelli, N. R. Jennings, and M. Wooldridge. Developing multiagent systems: The gaia methodology. *ACM Transactions on software Engineering and Methodology*, 12(3):317–370, 2003.