Agent-oriented modeling by interleaving formal and informal specification

A. Perini[1], M. Pistore[2,1], M. Roveri[1], and A. Susi[1]

[1] ITC-irst, Via Sommarive, 18, I-38050 Trento-Povo, Italy
{perini,roveri,susi}@irst.itc.it
[2] Department of Information and Communication Technology
University of Trento, via Sommarive 14, I-38050 Trento-Povo, Italy
pistore@dit.unitn.it

Abstract. The goal of this paper is to discuss possibilities of inter-mixing formal and informal specification in order to guide and support the conceptual modeling process in software development. We sketch a framework which rests on an agent-oriented methodology that provides a modeling language which allows for the definition of both informal and formal specification. We show how formal techniques can be used to guide and support the analyst while building and refining a conceptual model. Examples of its applications are discussed, with reference to the decision making process undertaken by the analyst when performing a set of activities relevant for requirements engineering, such as requirements elicitation and refinement, user validation of requirements specification, or management of requirements evolution. A case study taken from a technology transfer project in the agricultural domain is used to illustrate the approach.

1 Introduction

In the last years, a considerable effort in defining agent-oriented approaches to software development is going on [6, 12, 22]. The main reason for this can be drawn back to the recognition that the agent paradigm, beside providing a usefull technology to build software systems with an open architecture, offers appropriate abstractions for specifying and designing critical properties of these system, such as the dynamic evolution of their architecture and the interaction protocols of system components.

Most of the proposed methodologies adopt visual modeling as a core process, according to a best practice in structured Object Oriented software development processes [14], as well as to ideas proposed by Agile software development approaches [1].

The usage of visual modeling languages in the software development process offers advantages such as that of providing an effective communication framework for different stakeholders of the process. For instance, a use-case model can be used to discuss system requirements with the users. Nevertheless, visual modeling languages which lack a formal definition of their semantic, can lead to subjective models, which can hardly be refined in a straightforward way into a system design. Moreover, a typical question when building a conceptual model is: “when can I stop refining it?” This weakness is usually addressed by structured methodologies, which provides guidelines

for the analyst and for the designer in building, refining and documenting the process' artifacts that are based on conceptual models, e.g., [14].

Formal specification languages solve some of the weaknesses of visual modeling languages, specifically, they permit to define models with a precise semantics, and facilitate their transformation into system designs. However, writing formal specification languages usually require strong skills, and formal specifications are often very ineffective for discussing with the stakeholders. Moreover, the formalization "a posteriori" of the visual languages provided by the conceptual modeling frameworks is not an easy task at all, due to the ambiguities in the meaning of the graphical notations.

The basic idea discussed in this paper concerns the possibility to exploit formal techniques to guide and support the analyst while building and refining a conceptual model. In particular we will focus on the decision making process undertaken by the analyst when performing a set of activities, relevant for requirements engineering, such as requirements elicitation and refinement, or user validation of requirements specification. We will also analyze how results of deductive reasoning procedures run on a formal specification can support the analyst's decisions. Our claim is that, if the diagrammatic models are equipped with a formal semantics, then only a limited formalization effort is necessary to exploit formal techniques. Moreover, the diagrammatic notations make it possible to interpret the models in an informal way, for instance when discussing with the stakeholders.

In our approach we refer to the *Tropos* methodology [3, 18], an agent oriented software development methodology which provides a conceptual modeling language that can be used both to build an informal specification or a formal one. It is this common conceptual model that allows for the formal interpretation of the diagrammatic models that we have described above. Our long term objective is that of providing a tool that supports the analyst and the designer that use informal modeling, in performing the deductive reasoning on a formal specification which has been automatically derived from the informal model.

The paper is structured as follows. Section 2 recalls the basic features of conceptual modeling in *Tropos* and how to build informal and formal specification. Section 3, presents our approach to interleaving informal and formal specification with reference to specific requirements engineering activities. Related works are discussed in Section 4. Finally, conclusions are presented in Section 5.

2 Background

The *Tropos* methodology [3, 18] is an agent-oriented software development methodology which provides a visual modeling language that can be used to define both an informal specification and a formal one. ¿From a practical point of view, the methodology guides the software engineer in building an informal, conceptual model that is incrementally refined and extended from an early requirements model, namely a representation of the organizational setting where the system-to-be will be introduced, to system design artifacts, according to a requirements-driven approach.

The *Tropos* language allows to model intentional and social concepts, such as those of actor and goal, and set of relationships, such as actor dependency, goal decomposi-

tion, means-end and contribution relationships. These elements support the modeling of basic goal analysis techniques. The language ontology has been given in terms of common sense (informal) definitions. An *actor* models an entity that has strategic goals and intentionality, such as a physical agent, a role with respect to a given context, or a set of roles (i.e., a position). *Goals* represent the strategic interests of actors. Two basic type of goals are considered, namely hard and soft goals, the latter having no clear-cut definition and/or criteria as to whether they are satisfied. Softgoals are useful for modeling goal/plan qualities and non functional requirements. A *dependency* between two actors indicates that an actor depends on another in order to achieve a goal, execute a plan, or exploit a resource.

Basic modeling activities in *Tropos* include the identification of the actors with their goals and of the actors mutual dependencies. Each goal can be analyzed from the point of view of the individual actor considering: possible sub-goals (*AND decomposition*); means to satisfy these goals (*means-end relationship*); alternative ways to achieve a specific goal (*OR decomposition*); goals or plans or resources that can contribute positively or negatively to its achievement (*contribution*). All these models can be depicted using two basic types of diagrams, namely, actor and goal diagrams. A detailed account of modeling activities can be found in [3].

A *Tropos* specification provides a "static" view of the organizational setting and of the dependencies among the different elements of the domain. A Formal *Tropos* (*FT* hereafter on) specification [9, 11] extends a *Tropos* specification with annotations that characterize the valid behaviors of the model. In *FT* the emphasis goes in modeling the "strategic" aspects of the evolutions of the model. Thus, an *FT* specification consists of a sequence of class declarations such as entities, actors, goals, and dependencies. These are the formal counterparts of the elements of the "informal" *Tropos* specification. Each declaration associates a set of attributes to the class and characterizes its instances. Moreover, class declarations contain temporal constraints expressed in a typed first-order linear time temporal logic (LTL). These constraints describe the valid lifetime evolutions of the model in terms of temporal evolutions of set of instances of the classes in the specification. Two critical moments in the life-cycle of goals and dependencies are the instants of their *creation* and *fulfillment*. The creation of a goal is interpreted as the moment in which the owner or depender expects or desires to achieve the goal, while its fulfillment is the moment in which the goal condition is actually achieved. In *FT*, creation and fulfillment constraints can be used to define conditions for these two moments in the life of intentional elements. Creation and fulfillment conditions are used, e.g., for defining constraints on the lifetimes of sub-goals in a goal decomposition (sub-goals are created after the parent goal and should be fulfilled before the parent goal can be fulfilled), or for defining the responsiveness of an actor w.r.t. the dependencies (an actor can take care immediately of some of them while delaying other dependencies). *FT* also provides *invariant* constraints that define conditions that should be true throughout the lifetime of class instances. Typically, invariants define relations on the possible values of attributes, or cardinality constraints on the instances of a given class.

Given an *FT* specification, one can ask questions such as: Can we construct valid operational scenarios based on the model? Is it possible to fulfill the primary goals of

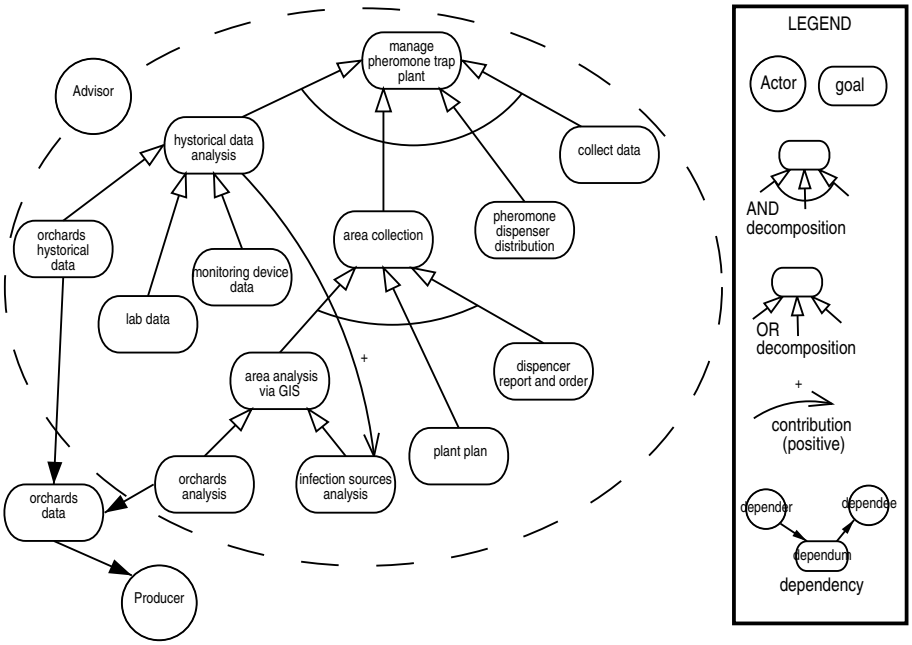

Fig. 1. Goal diagram of the actor **Advisor**, showing an example of goal analysis.

actors in the current model? Do the decomposition links induce a meaningful temporal order for goal fulfillment? Do the dependencies represent a valid synergy or synchronization between actors? These questions can be formulated as formal queries on a *FT* model and answered by the T-TOOL [10], an automatic verification tool based on the NUSMV [7] model checker (see [9, 10] for more details).

The effectiveness of the *Tropos* (and of the *FT*) methodology has been illustrated by several case studies [10, 11, 18, 19]. In the following we will introduce a simple example giving both the informal specification and the formal one. The example is extracted from a real case-study developed in a technology transfer project in the domain of decision-support systems in agriculture, described in [19]. In particular, we will focus on goal modeling, represented by a simple goal diagram, and we will describe basic questions that the analyst can issue during informal modeling and that can be automatically answered when adding formal annotations.

2.1 Informal modeling

The example considered corresponds to a fragment of the early requirements model for the agriculture domain. The early requirements model in *Tropos* describes the domain stakeholders (modeled as actors), their goals, and the mutual dependencies. In our case, the actor **Producer** represents the apple grower who pursues objectives such as apply-

ing Integrated Production (IP) [3] techniques with the help of agronomists of the local advisory service, represented by the actor **Advisor**. The example focuses on a particular technique for reducing the risk of infection of an apple pest which requires to install in the orchard a pheromones trapping system which prevents the pest population growth in the area. The design of the trapping system needs to take into account the geometry of the field (e.g., perimeter), geographical feature of the area of the field and the possible infections sources in the neighborhoods.

Figure 1, shows goal analysis of the **Advisor**, as resulting from a set of interviews to agronomists, concerning approaches currently in use for applying pheromones trapping techniques.

During goal analysis the analyst intends to extract all the possible steps that the **Advisor** has to fullfil to achieve the higher level goal **manage pheromone trap plant**. The analysis starts from the decomposition of this goal **manage pheromone trap plant** in a set of goals that has to be all satisfied (AND decomposition). An important step is the analysis of the agronomical history of the area of interest (modeled with the goal **hystorical data analysis**), that can be done by finding the data directly from the orchards descriptions (**orchards hystorical data**) maintained by the producers, or by accessing the data from the research laboratory (**lab data**), or by using data coming from the orchard monitoring systems information, like meteorological data, (see the goal **monitoring devices data**). In this case the OR decomposition allows the analyst to model alternative ways to access data that can be available from different information sources. Crucial for the advisor is the possibility to collect information about geographical and biological charcteristics for the areas that seems to be candidate for the application of the pheromone trapping techniques (modeled with **area collection**). This goal can be accomplished sub-goals, namely, the geographic analysis of the areas (**area analysis via GIS**), the planning of the pheromone system (**plant plan**) and the distribution of the pheromone dispensers (**dispenser report and order**). All these goals have to be satisfied in order to obtain the set of needed information on the area.

The dispenser distribution and the monitoring of the results obtained upon application of the techniques complete the accomplishment of the higher level goal.

2.2 Formal Modeling

An excerpt of the *FT* specification for the agriculture example is depcited in Figure 2. The *FT* specification can be obtained from the *Tropos* model by mapping actors and intentional elements into corresponding *FT* classes. The attributes in *FT* are references to other classes. For example, goal **OrchardAnalysis** refers to the **HistoricalDataAnalisys** goal that motivates the advisor to get the historical data (attribute **hda**). Similarly, dependency **OrchardsData** refers to **OrchardHystoricalData** and to **OrchardAnalysis** goals of the advisor (attributes **ohd** and **oa** respectively). Since actor **Advisor** is the owner of goals **ManagePheromoneTrapPlant**, **AreaCollection** and **OrchardAnalysis**, the *FT* specification has **Advisor** as the Actor attribute of the three

[3] Integrated Production (IP) aims at a sustainable approach to agriculture production. In plant disease control, it promotes the use of low-impact techniques and chemicals, and the exploitation of natural defense mechanisms.

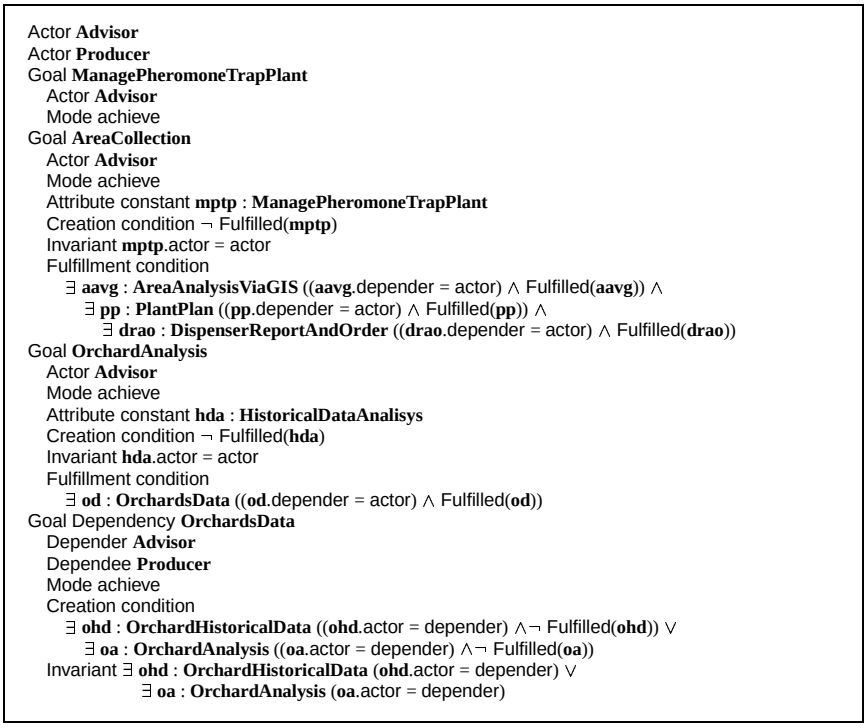

Fig. 2. Excerpt of *FT* specification.

goals. Similarly, Depender and Dependee attributes of dependencies represent the two parties involved in a delegation relationship. Goals and dependencies in Figure 2 are also equipped with a mode attribute, which defines the modality of fulfillment. The mode of goal **ManagePheromoneTrapPlant**, for instance, is achieve, which means that the advisor wants to reach a state where he was able to manage the pheromone trap plant, and therefore the goal is fulfilled. Another modality in maintain, where the fulfillment condition is to be continuously maintained. Figure 2 contains also some examples of constraints on the lifetime of class instances. For instance, the first invariant of Figure 2 binds a **AreaCollection** object with its father **ManagePheromoneTrapPlant** object. Typically, primary intentional elements, (e.g., **ManagePheromoneTrapPlant**) have fulfillment constraints, but no creation constraints: we are not interested in modeling the reasons why an advisor wants to manage a pheromone trap plant. Subordinate intentional elements (e.g., **AreaCollection**, **OrchardAnalysis**) typically have constraints that relate their creation with the state of their parent intentional elements. For instance, Figure 2 shows that a creation condition for an instance of goal **AreaCollection** is that the parent goal **ManagePheromoneTrapPlant** is not yet fulfilled: if the advisor has already managed the pheromone trap plant there is no need to manage it again (unless a new management activity is started). The creation condition of dependency **OrchardsData** together with the fulfillment condition of goals **Orchard-**

HystoricalData and **OrchardAnalysis** elaborate the delegation relationship between **Advisor** and **Producer** in the corresponding *Tropos* diagram.

In an *FT* specification we can also specify properties desired to hold in the domain, so they can be verified against the model we built. In *FT* we distinguish between assertion properties that should hold for all valid evolutions of the *FT* specification, and possibility properties that should hold for at least one. Given the *FT* specification and a set of properties, we can verify whether the *FT* specification satisfies the properties by means of T-TOOL.

3 The framework

The proposed framework is based on the idea of exploiting formal techniques to guide and support the analyst while building and refining a conceptual model. We describe it focusing on a set of requirements engineering activities, such as requirements elicitation and refinement, or user validation of requirements specification. These activities involve domain stakeholders and an informal analyst (called from now on iAnalyst), which can pose specific questions to a formal analyst (fAnalyst)[4]. After a preliminary acquisition of information on domain stakeholders, on their goals and on their reciprocal dependencies, the iAnalyst starts building an early requirements model, such as that described by the diagram depicted in Figure 1.

Once the preliminary early requirements model has been devised, the iAnalyst proceeds with the analysis of this model. Covered activities are: the *assessment of the model* against possible inadequacies, incompleteness or inconsistencies; the *validation of the resulting specification* with the stakeholders in order to end up with an agreed set of requirements; and, when inconsistencies are discovered the needs of model revision, the *management of model refinement and evolution*, in order to maintain its consistency, propagate changes, merge redundant information.

In these activities, the iAnalyst has the possibility to make queries on specific aspects of the model that the fAnalyst translates into *FT* properties and checks of the *FT* model. The queries of the iAnalyst may concern different aspects of the model. In particular, the iAnalyst can check:

- the capability of an actor to fulfill a given goal, possibly assuming that some subgoals or some dependencies cannot be achieved;
- the presence of constraints on the temporal order in which activities/goals can be started and/or achieved;
- the presence of implicit cardinality constraints on the relationships among goals in a goal diagram;
- the fact that a given property is respected by all the valid behaviors of the model;
- the fact that a given property is exhibited by some of the valid behaviors of the model.

[4] Our long term goal is that of deriving the requirements of a CASE tool that could play the role of the fAnalyst, as emerging from the following discussion. For the moment, the role of the fAnalyst has to be played by a human actor.

In most of the cases, the formalization effort required to the fAnalyst for answering to these queries is small. The *FT* model can be obtained by an automatic translation of the informal model [10], and the *FT* property is obtained directly by the informal query of the iAnalyst.

In the following, we discuss three examples that illustrate the framework with respect to the different requirements engineering activities. In each example we identify a set of specific questions the iAnalyst can be interested in; we formalize these questions using a high level query language; we show how to map these queries into low-level *FT* properties that are checked by the T-TOOL; and we discuss the answers obtained for the queries and possible ways for visualizing them. We also comment on the benefits and costs of the proposed approach. For explanatory purposes, we use the simple goal diagram depicted in Figure 1. We remark that on this small model it is easy to answer to the questions of the iAnalyst with a direct inspection of the model. On larger models, the added value of using the fAnalyst services becomes more and more relevant.

3.1 Example 1. Assessing and refining the informal model

The iAnalyst assesses the model against possible inconsistencies or redundancies or critical elements, and eventually refines the informal model on the basis of the answer of the fAnalyst to his/her questions.

Given the goal diagram depicted in Figure 1 a possible question of the iAnalyst can be:

*Is it possible for the advisor to perform **historical data analysis**?*

We can formulate this question as the following query:

FULFILLABLE **HistoricalDataAnalysis**

This query corresponds to the the low level *FT* property:

Global Possibility
F (∃ **a** : **Advisor** (∃ **hda** : **HistoricalDataAnalysis** (**hda**.actor = **a** ∧ Fulfilled(**hda**))))

The *FT* specification admits a scenario that conforms with the above formula, and this scenario can be depicted in terms of a frame sequence, as in Figure 3. In the scenario,

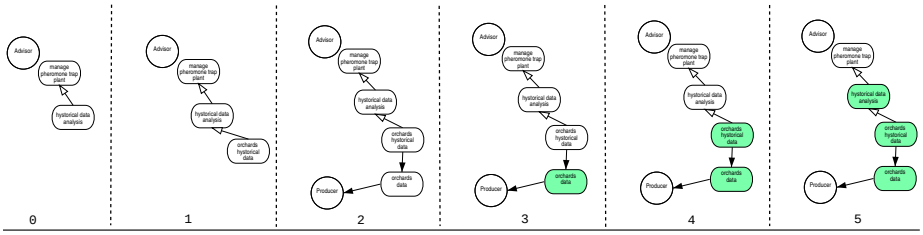

Fig. 3. A scenario where the **historical data analysis** goal is fulfilled.

historical data analysis is fulfilled using **orchard hystorical data** received from the **Producer**. Each frame corresponds to a step of the scenario instantiation process: the first frame shows the creation of the goal the query refers to, the second and third complete the creation of sub-goals till a goal delegation is reached; in the following three frames, the effects of the satisfaction of the delegated goal, shown by the dashed texture, is propagated back, along the goal decomposition till the goal under inspection. The above question can be cosidered a specific instance of the general question:

> *Focusing on a specific actor goal, does the current model allow to achieve it in some valid scenario?*

If the answer to such questions is positive, the result can be effectively shown by a diagram analogous to the one depicted in Figure 3. If a scenario does not exist, a warning message is emitted.

Another type of question the iAnalyst can pose when analyzing the advisor goal diagram is:

> *How critical is the dependency on the actor **Producer** in order to satisfy the main advisor's goal **manage pheromone trap plant**?*

This question can be reformulated as "*Is it possible to fulfill **manage pheromone trap plant** without fulfilling all the dependencies with the **Producer** actor?*" This question can be mapped to the following query expressed in the high level query language:

NONCRITICAL **Producer** FOR **ManagePheromoneTrapPlant**

which corresponds to the the low level query:

Global Possibility
F (∃ **a** : **Advisor** (
∃ **mftp** : **ManagePheromoneTrapPlant** (**mftp**.actor = **a** ∧ Fulfilled(**mftp**) ∧
∀ **p** : **Producer** (∀ **od** : **OrchardData** (**od**.depender = **a** ∧ **od**.dependee = **p** →
¬Fulfilled(**od**))))))

When this query is submitted to T-TOOL, a witness scenario is generated that conforms to the specification (see Figure 4). The above question can generalized by the following:

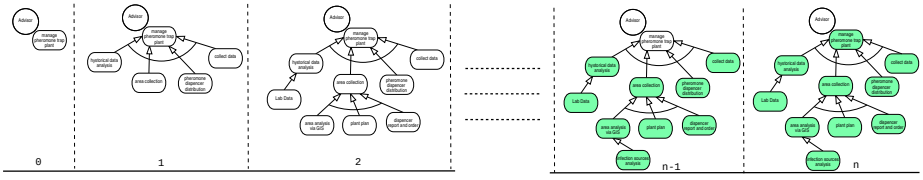

Fig. 4. A scenario where the **manage pheromone trap plant** goal is fulfilled without any dependency from a **Producer**.

Focusing on a specific actor, what are the critical dependencies on external actors? That is, what actors can prevent the achievement of a main goal if they do not achieve delegated goals?

This kind of questions can be seen as queries on the graph corresponding to a goal diagram. They are particularly useful when dealing with a complex model for which a direct inspection of the diagram is impractical. For these questions, the informal queries can be automatically translated into the relative *FT* specification, so no additional effort to the iAnalyst is required.

3.2 Example 2. Validating the informal model with the stakeholders

The iAnalyst looks for relevant validation cases to be proposed to the stakeholders in order to drive the validation process and to end up with an agreed model. Validation cases can be suggested by the fAnalyst on the basis of the model structure.

Given the actor diagram depicted in Figure 1 a possible question of the iAnalyst in this case is:

*How does the advisor usually operate? Is that he/she always satisfies the goal **area collection** after satisfying all its sub-goals, according to the following sequence: **area analysis via GIS**, **plant plan** and finally **dispenser report and order**?*

We can formulate this query as follows:

FULFILL **AreaAnalysisViaGIS** THEN **PlantPlan** THEN **DispenserReportAndOrder** THEN **AreaCollection**

This corresponds to the the low level query:

Global Assertion F (
∀ **a** : **Advisor** (∀ **ac** : **AreaCollection** (**ac**.actor = **a** →
∀ **aavg** : **AreaAnalysisViaGIS** (**aavg**.**ac** = **ac** →
∀ **pp** : **PlantPlan** (**pp**.**ac** = **ac** →
∀ **drao** : **DispenserReportAndOrder** (**drao**.**ac** = **ac** →
(Fulfilled(**ac**) ∧ Fulfilled(**aavg**) ∧ Fulfilled(**pp**) ∧Fulfilled(**drao**)) →
P (JustFulfilled(**aavg**) ∧ X F (JustFulfilled(**pp**) ∧
X F (JustFulfilled(**drao**) ∧ X F JustFulfilled(**ac**)))))))))

When this question is submitted to T-TOOL, a scenario that violates the assertion is found. Figure 5 illustrates this scenario with a bar-charts diagram that shows the ordering in which the goals involved in the specification are created (beginning of the light bar) and fulfilled (beginning of the dark bar). The scenario shows a case where goal **dispenser report and order** is achieved before **plant plan**. This scenario can be discussed with the stakeholder, in order to understand whether this order of achievement is possible in the application domain. If it is possible, then the query can be refined in order to take into account that **dispenser report and order** and **plant plan** can be achieved in an arbitrary order. If it is not possible, then a temporal constraint between these two goals has to be added in the *FT* model, so that the invalid behavior is discarded. The above question can be cosidered a specific instance of the general question:

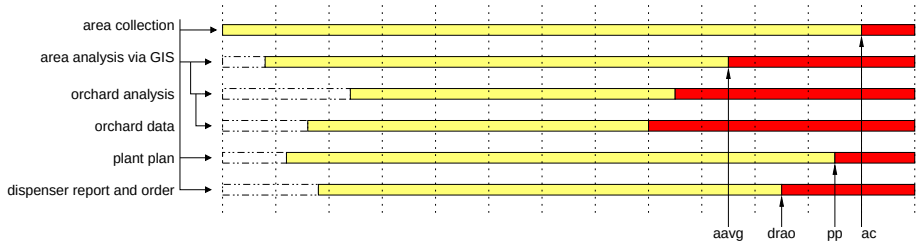

Fig. 5. A validating scenario where the goal **area collection** is fulfilled.

Focusing on a specific actor, is there an implicit temporal ordering in a goal decomposition, that should be explicited?

This service requires to store all possible ordering of the goals under consideration (a set of bar-chart diagrams like the one of Figure 5 can be used to this purpose) and to annotate those orders that the stakeholder has considered valid.

Another specific question that can be asked for when performing the analysis considered in this second example, is:

*In the case more than one **plant plan** activities can be performed before a **dispenser report and order**, is it always the case that all instances of **plant plan** relative to the same orchard have to be fulfilled before fulfilling the **dispenser report and order** for the orchard?*

We can formulate this query as follows:

FULFILL ALL **PlantPlan** BEFORE **DispenserReportAndOrder**

which maps to the *FT* assertion:

Global Assertion F (
∀ **a** : **Advisor** (∀ **ac** : **AreaCollection** (**ac**.actor = **a** →
∀ **drao** : **DispenserReportAndOrder** (**drao.ac** = **ac** →
Justfulfilled(**ac**) → ∀ **pp** : **PlantPlan** (**pp.ac** = **ac** → Fulfilled(**pp**))))))

Also in this case, T-TOOL generates a counter-example scenario, illustrated by the bar-charts diagrams depicted in Figure 6. The scenario shows that it is possible to fulfill the second instance of **plant plan** after the **dispenser report and order** has already been fulfilled. The question can be considered a particular case of the more general one:

Focusing on a specific actor, is there an implicit cardinality on the relationships which models a given goal decomposition that should be explicited? How do these relationships impact in the temporal ordering of the goal decomposition?

The formalization effort for the queries on implicit temporal orderings and on implicit cardinalities is higher than that of the basic queries seen for Example 1. Indeed, annotations have to be added to the *FT* model in order to implement the constraints on the temporal orders that are agreed with the stakeholder. On the other hand, the understanding of these constraints is very important for a deep understanding of the whole application domain, and their elicitation is very difficult in a purely informal framework.

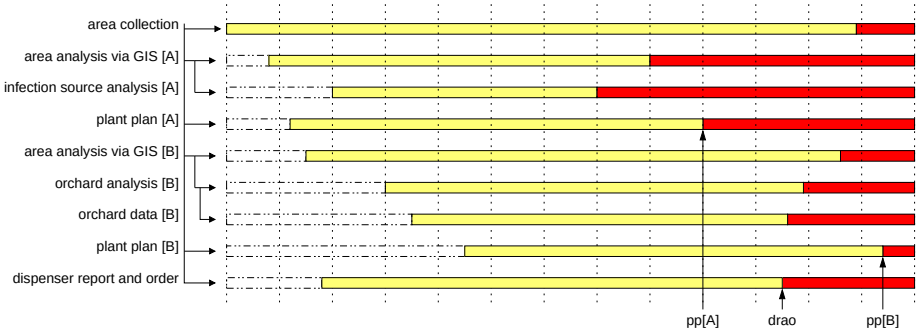

Fig. 6. Another validation scenario for the goal **area collection** with two instances.

3.3 Example 3. Managing the model evolution

The iAnalyst can exploit the fAnalyst services also to manage the model evolution, catching the creation of inconsistency due to a modification of a (previously consistent) model. In particular, question of the following type could be posed by the iAnalyst:

> *We remove **infection sources analysis**. Does this lead to critical dependencies from other actors?*

This corresponds to general questions on whether a change in the model makes it impossible to achieve a goal that was previously possible to achieve, or if it introduces new critical goals or dependencies. This question can be solved by re-executing the queries performed in previous steps. If some of the queries fail, than the result produced can be used to correct the specification. If they all succeed, than we can formulate additional queries to validate the specification as described previously.

Assume that in **area collection** we mark that **plant plan** has to be satisfied prior to **dispenser report and order** (as it is the case after a previous query). We now restructure the model, moving goal **dispenser report and order** one level up (see Figure 7). A second question could be:

> *Is the temporal order previously described for the goals **plant plan** and **dispenser report and order** still respected in the new model?*

The question can be seen as a request of advise on possible violation of previuosly validated temporal constraints, when some change in the AND/OR structure of goals has been introduced. In the specific example, the reconfiguration of the model has invalidated the constraint on the temporal order of the two goals, and a counter-example scenario is returned by the T-TOOL. In order to guarantee that **plant plan** is satisfied prior to **dispenser report and order**, a new temporal constraint has to be added to the model, for instance between goals **area collection** and **dispenser report and order**.

We remark that the possibility of re-running the queries when the model changes is one of the most important benefits of the approach that we are proposing in the paper. Indeed, it allows to identify the effects of the changes to the properties that the model is supposed to satisfy as soon as the changes are performed on the model.

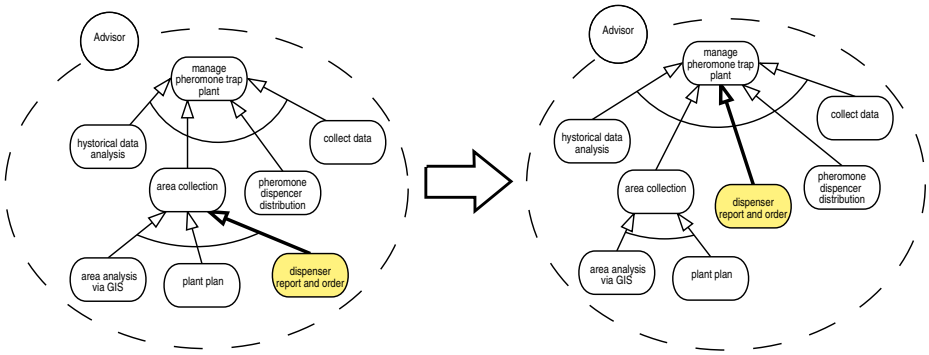

Fig. 7. The modified goal diagram if temporal ordering constraints are taken into account.

4 Related work

Different lines of research are relevant to the work presented here, as briefly mentioned in the following. Close relationships of *Tropos* with other Agent-Oriented Software Engineering (AOSE) methodologies, such as Gaia [21], MaSE [8], Prometheus [17] and AUML [2] based methodologies, can be found, as discussed in [3]. Among works relevant to the *FT* research we shall mention *KAOS* [15], a framework that supports requirements analysis adopting a goal- and agent-oriented approach, and the *Topoi diagrams* [16]. Topoi diagrams are related to *Tropos* diagrams, where intentional links describe influences between the intentional elements of a domain. A detailed discussion of their relationships with *FT* is presented in [10].

The importance of Formal Methods is widely recognized in the AOSE community, and there are several approaches to AOSE that are based on formal techniques. See [5] for a presentation of some of these techniques and of their application to the specification, to the verification, as well as to the automated generation of an implementation of agent systems. Most of these approaches, however, require a heavy formalization effort and strong skills in order to be used effectlively. Our aim is to allow for a lightweight usage of formal verification techniques, that are used as services in an "informal" development methodology. A recent approach, called ATOS [13], adopts a similar approach with regard to the detailed design of interaction protocols in Multi-Agent System design. ATOS introduces a textual notation of AUML that can be translated to an extended finite state machine which can be processed by a model checker. ATOS has been exploited to perform formal verification of AUML sequence diagram specification of interaction protocols of Multi-Agent System.

A deep analysis of the importance of the requirements engineering activities considered in our approach is presented in [20], where specific tasks that could benefit from a formal specification approach (even if costs in development and assessment are still considered high) are pointed out. We agree with the following general consideration borrowed from this work, which states that: "the by-products of a formal specification process are often more important than the formal specification itself, including a better

informal specification, obtained by feedback from formal expression, structuring and analysis".

More generally, approaches aiming at integrating knowledge engineering techniques with software engineering are worth to be mentioned.In [4] we can found an analysis of decision making in software development and maintenance, such as, planning and scheduling activities of component development and integration (e.g., deciding an optimal integration test order), assessing costs and benefits of introducing a given inspection technique during the development process. These problems have been reformulated as optimization problems for which approches exploiting techniques such as genetic algorithm and heuristic have been proposed. We are addressing decision making in a different set of activities, namely specification and conceptual modeling.

5 Conclusion and Future Work

This paper described a lightweight usage of formal verification techniques when performing conceptual modeling within an agent-oriented methodology which provides a modeling language that can be used both to build an informal specification or a formal one [3, 10].

A preliminary analysis of the proposed framework has been discussed with reference to a set of activities, relevant for requirements engineering, such as requirements elicitation and refinement, user validation of requirements specification, or management of requirements evolution.We considered the decision making process of the analyst when performing those activities and we discussed how it can be supported by formal verification services. Along this line we are defining additional verification services. Moreover, this approach will be extended to other activities in the software development process, such as architectural and detailed design.

An ultimate objective, beside that of providing the automatic translation of an informal model to a formal one, is that of developing a tool that supports the analyst and the designer which use informal modeling, for performing the deductive reasoning on a formal specification, by formulating queries analogous to those discussed in the examples of this paper.

References

1. Scott W. Ambler. Agile modeling essays, 2003. http://www.agilemodeling.com/essays.htm.
2. B. Bauer, J. P. Müller, and J. Odell. Agent UML: A formalism for specifying multiagent software systems. *Int. Journal of Software Engineering and Knowledge Engineering*, 11(3):207–230, 2001.
3. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An Agent-Oriented Software Development Methodology. *Journal of Autonomous Agent and Multi-Agent Systems*, 2003. To appear.
4. L. C. Briand. On the many ways software engineering can benefit from knowledge engineering. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 3–6, Ischia, Italy, july 2002. ACM.

5. P. Ciancarini and M. Wooldridge. Agent-oriented software engineering: the State of the Art. In *Agent-Oriented Software Engineering, First International Workshop, AOSE 2000*, number 1957 in LNCS, pages 1–28, Limerick, Ireland, jun 2000.
6. P. Ciancarini and M. Wooldridge, editors. *Agent-Oriented Software Engineering*, volume 1957 of *Lecture Notes in AI*. Springer-Verlag, March 2001.
7. A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NUSMV 2: An opensource tool for symbolic model checking. In *Computer Aided Verification*, number 2404 in LNCS, Copenhagen (DK), July 2002. Springer.
8. S. A. Deloach. Analysis and Design using MaSE and agentTool. In *12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2001)*, Miami University, Oxford, Ohio, March 31 - April 1 2001.
9. A. Fuxman. Formal analysis of early requirements specifications. Master's thesis, University of Toronto, 2001.
10. A. Fuxman, L. Liu, M. Pistore, M. Roveri, and J. Mylopoulos. Specifying and analyzing early requirements: Some experimental results. In *IEEE Int. Symposium on Requirements Engineering*, Monterey (USA), September 2003. IEEE Computer Society.
11. A. Fuxman, M. Pistore, J. Mylopoulos, and P. Traverso. Model checking early requirements specifications in Tropos. In *IEEE Int. Symposium on Requirements Engineering*, pages 174–181, Toronto (CA), August 2001. IEEE Computer Society.
12. F. Giunchiglia, J. Odell, and G. Weiß, editors. *Agent-Oriented Software Engineering III*. LNCS. Springer-Verlag, Bologna, Italy, Third International Workshop, AOSE2002 edition, July 2002.
13. J. L. Koning and I. Romero-Hernandez. Generating machine processable representations of textual representations of auml. In *Agent-Oriented Software Engineering III, Third International Workshop, AOSE 2002*, number 2585 in LNCS, pages 126–137. Springer, july 2002.
14. Philippe Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley, 2nd edition, 2000.
15. E. Leiter. *Reasoning about Agents in Goal-oriented Requirements Engineering*. PhD thesis, Universite Catholique de Louvain, 2001.
16. T. Menzies, J. Powell, and M. E. Houle. Fast formal analysis of requirements via "topoi diagrams". In *the* 23^{rd} *Int. Conference on Software Engineering*, pages 391–400, Toronto, CA, May 2001. ACM Press.
17. L. Padgham and M. Winikoff. Prometheus: A Methodology for Developing Intelligent Agents. In Giunchiglia et al. [12].
18. A. Perini, P. Bresciani, F. Giunchiglia, P. Giorgini, and J. Mylopoulos. A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. In *Proceedings of Agents 2001*, Montreal CA, May 2001. ACM.
19. A. Perini and A. Susi. Designing a Decision Support System for Integrated Production in Agriculture. An Agent-Oriented approach. *Environmental Modelling and Software Journal*, 2003. to appear.
20. Axel van Lamsweerde. Formal specification: a roadmap. In *ICSE 2000, 22nd International Conference on on Software Engineering, Future of Software Engineering Track*, pages 147–159, Limerick Ireland, June 2000. ACM.
21. M. Wooldridge, N. R. Jennings, and D. Kinny. The GAIA Methodology for Agent-Oriented Analysis and Design. *Journal of Autonomous Agents and Multi-Agent System*, 2000.
22. M.J. Wooldridge, G. Weiß, and P. Ciancarini, editors. *Agent-Oriented Software Engineering II*. LNCS 2222. Springer-Verlag, Montreal, Canada, Second International Workshop, AOSE2001 edition, May 2001.