Tool-supported Development with Tropos: The Conference Management System Case Study.

M. Morandini, D. C. Nguyen, A. Perini, A. Siena, and A. Susi

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

Abstract. The agent-oriented software engineering methodology *Tropos* offers a structured development process and supporting tools for developing complex, distributed systems.
The objective of this paper is twofold: first, to illustrate the use of *Tropos* to develop a Multi-Agent System, performing basic analysis and design activities, code generation and testing, with the support of a set of tools; second, to enable the comparison with other, tool-supported, agent-oriented software engineering methodologies through a description of the main steps of these activities and of excerpts of the resulting artefacts, with reference to a common case study, namely, the Conference Management System case study.

1 Introduction

Many Agent-Oriented Software Engineering (AOSE) methodologies have been proposed over the last years [13, 7]. This fact motivated research on how to compare and evaluate these methodologies, with the purpose of pointing out differences and complementarities, and of giving criteria for selecting the most appropriate methodology, for a given development scenario [13, 5].

While this research field is becoming more mature, a need is emerging for detailed guidelines when applying a methodology along core phases in the software development process, and for supporting tools. This is considered a crucial step towards the adoption of AOSE methodology by industry.

The *Tropos* methodology, proposed in [3], is an agent-oriented methodology for developing complex, distributed systems. A peculiarity of *Tropos* is that it adopts a requirement driven approach to software development, recognizing a pivotal role to the modelling of domain stakeholders and to the analysis of their goals, before generating a design for the system-to-be. System design then consists in specifying software agents who have their own goals and capabilities that are intended to support the fulfilment of stakeholder goals.

Further research on the *Tropos* methodology focused on its application in developing specific classes of applications, as for instance distributed knowledge management systems [23]. Moreover, extensions of its modelling language have been proposed to support the analysis of crucial issues in distributed systems, such as trust and security [10]. Several tools have been built as well. *TAOM4e*, for supporting a model-driven, agent-oriented approach to software development [19, 17], the T-Tool [9], for performing model-checking of *Tropos* specifications, the GR-Tool for supporting formal reasoning on goal models [12], multi-agent planning for supporting the selection among alternative networks of delegations [4].

The main objectives of this paper are: first, to illustrate how to use *Tropos* to develop a Multi-Agent System (MAS), performing basic analysis and design activities, generating code and performing testing on it, with the support of a set of tools; second, to enable the comparison with other tool-supported AOSE methodologies through a description of the main steps of these activities and of excerpts of the resulting artefacts, with reference to a common case study, namely, the Conference Management System (CMS) case study [6].

The paper is structured as follows. Section 2 recalls basic development activities in *Tropos* and gives a short description of the tools that support them. Requirements analysis is described in Section 3, system design is described in Section 4, code generation and testing in Section 5. Considerations emerged during the development of the CMS case study are discussed in Section 6. Finally, conclusion and future work are presented in Section 7.

2 Tropos Development Process and Tools

The software development process in *Tropos* is structured in five main phases, namely: early requirements analysis that focuses on the understanding of the existing organizational setting where the system-to-be will be introduced; late requirements that deals with the analysis of the system-to-be; architectural design that defines the system's global architecture in terms of subsystems; detailed design that specifies the system agents micro-level; implementation that concerns code generation according to the detailed design specifications.

This development process is model-based, that is, requirements and design models are core artefacts. They are built using a conceptual modelling language, derived from the i* framework [24]. This modelling activity, called Agent-Oriented (AO) modelling in Fig. 1, spans the first four phases in the software development process. Basic concepts of the modelling language are those of actor, goal, plan and dependency for goal achievement. [1] AO modelling can be performed using the *TAOM4e* modelling tool [19]. The tool has been extended to support automatic code generation from the *Tropos* specification into *JADE* [1] or *Jadex* [20] MAS, by exploiting a mapping between the *Tropos* meta-model concepts and the target implementation languages constructs [18, 15].

Other tool-supported analysis techniques are available in *Tropos*, such as validation of requirements specification via model-checking (see T-Tool [9]) or formal analysis on goal models of requirements and system design (see GR-

[1] UML activity and sequence diagrams may be used as well for detail design in *Tropos*.

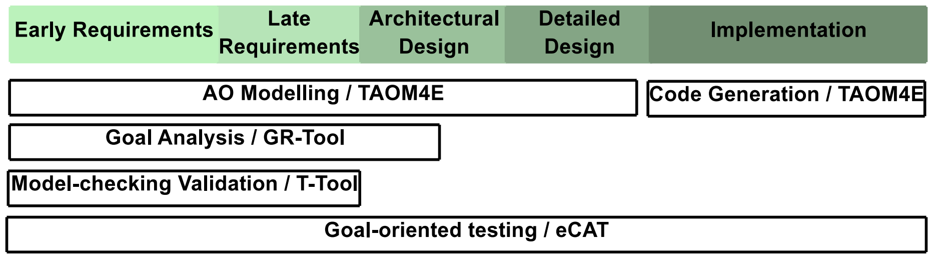


Fig. 1. Development Process Phases: Activities and Supporting Tools.

Tool [12]). These types of analysis are particularly useful in case of complex models. They will be not described in this paper.

Goal-Oriented testing has been recently proposed as a complementary activity to AO modelling and code generation activities [16]. The basic idea is that of deriving test cases directly from the AO specifications produced along the development process with the aim to support testing and validation along the process phases. In this paper we will illustrate agent and integration testing as supported by the eCAT tool [16].

A high-level architecture of the tooled environment is described below, while the use of the tools during the development of the CMS system will be illustrated in the following sections.

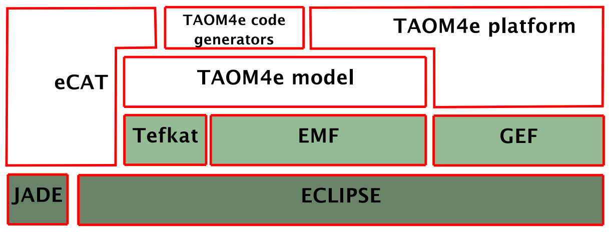


Fig. 2. Architecture of *TAOM4e* and *eCAT*.

2.1 TAOM4e and Code Generation functions

TAOM4e [2] is a graphical *Tropos* modelling framework, which supports modelling in all phases of the *Tropos* process. It is realized as a plug-in for the Eclipse [3] project and extends existing plug-ins, as shown in Fig. 2: the EMF plug-in[4] offers a modelling framework and code generation facility for building tools and other

[2] `http://sra.itc.it/tools/taom4e`
[3] `http://www.eclipse.org`
[4] `http://www.eclipse.org/emf`

applications based on a model specification described in XMI; the Graphical Editing Framework (GEF) plug-in[5] allows to create a graphical editor from an existing application model; the Tefkat plug-in[6] provides a rule-based language to implement model to model transformation.

The *Tropos* metamodel has been implemented on top of EMF (*TAOM4e* model), GEF is used to realize the graphical representation of the model and the different views on it (*TAOM4e* platform), whereas the Tefkat plug-in is used to transform top-level plans and their decompositions into UML activity diagrams. The resulting diagrams can be edited using any UML2 editor and further detailed with sequence diagrams, which define communication protocols among agents.

Fig. 3 shows a screen-shot of the *TAOM4e* GUI.

The *TAOM4e* modeller is enriched with *TAOM4e* generators to derive skeletons of code for the *JADE* and *Jadex* agent platforms, directly from an UML specification of detailed design artefacts or from a *Tropos* goal model. *TAOM4e* generators include *UML2JADE*, *t2x*, and *Tropos2UML*. *Tropos2UML* can be used to generate UML activities diagrams from *Tropos* goal model, while *UML2JADE* can generate *JADE* agent code from UML activity and sequence diagrams that specify *Tropos* plans (capabilities), details are given in [17].

The part of an agent that is responsible for choosing the right plans at run-time in order to reach the desired goals is called *knowledge level*. In an agent's *GM*, the *knowledge level* consists of goals and their decomposition, contributions, dependencies to other agents and means-end relations to plans. These are inputs for the *t2x* (namely *Tropos* to *Jadex*) tool. The tool generates skeletons for agents following the *BDI architecture*, and they are executable on the Jade BDI agent platform [21]. The mapping between *Tropos* goal model elements and *Jadex* construct is described in [18, 14].

The generated code skeleton implements the reasoning part of a software agent. It consists of an *Agent Definition File* (ADF), in XML format, which defines goals, plans, beliefs and messages for every system agent in the *GM*. The single plans can be implemented in Java files, which can be associated to the elements in the ADF.

2.2 eCAT

eCAT [7] implements our method for automated continuous testing of MAS, supporting a goal-oriented testing approach [16]. The tool facilitates test suites derivation from goals analysis and generates semi-automatically test suites from goal analysis diagrams produced with *TAOM4e*. It also provides GUIs to help human testers specifying test inputs and oracles. Moreover, *eCAT* can evolve and generate more test inputs during the course of testing, and run these test

[5] `http://www.eclipse.org/gef`
[6] `http://tefkat.sourceforge.net`
[7] `http://sra.itc.it/people/cunduy/ecat`

inputs continuously to test the MAS. In this way, the MAS under test is tested more thoroughly and is stressed more extensively.

eCAT consists of three main components: *Test Suite Editor*, *Autonomous Tester Agent*, and *Monitoring Agents*. Its operation is described as follows:

– Based on agent specifications and design (e.g. outputs of AO modelling with *TAOM4e*), the *Test Suite Editor* generates initial test suites and then provides a GUI for end-users to edit them.
– The *Autonomous Tester Agent* takes those test suites and/or generates other test suites randomly. It then continuously executes them against the multi-agent system under test. During the course of test execution, the *Autonomous Tester Agent* can evolve test suites by applying a mutation and evolutionary technique in order to create more test suites, which aim at revealing more bugs.
– The *Monitoring Agents* assist the *Autonomous Tester Agent* during testing. By monitoring events and interactions happened in the multi-agent system and its environment, it provides useful information to the *Autonomous Tester Agent* in order to judge if a test passes or fails, and a trace to the found bug when failed.

3 Requirements Analysis

Starting software development in *Tropos* using *TAOM4e* requires to create a "*Tropos* project" that will collect all the artefacts generated during the development process, such as models, actor and goal diagrams that represent views on these models, agent code, test cases and logs generated during test execution.

Two models are built in the requirements analysis phases: the **Early Requirements** and the **Late Requirements** models. They are in charge of describing the domain setting as is and the same domain once the system-to-be will have been introduced, respectively.

A guide to start building the Early Requirements model is given by the following analysis questions: *Who are the stakeholders in the domain? What are their goals and how are they related to each other? What are there strategic dependencies between actors for goal achievement?*

The Conference Management System domain is modelled in terms of its main stakeholders (actors), namely papers' authors, by the actor Author, the conference's program committee and its chair, by the PC and the PC Chair actors respectively, papers reviewers by the actor Reviewer and the proceedings publisher by the actor Publisher. Stakeholders' goals are then identified and, for every goal, the analyst can decide, on the basis of the domain documentation, if the goal is achievable by the actor itself or if the actor has to delegate it to another actor, revealing a dependency relationship between the two actors, such as in the case of the dependency between Author and PC for the achievement of the goal Publish proceedings. An analogous analysis can be carried on for the domain tasks and resources, according to the *Tropos* modelling process described in [11].

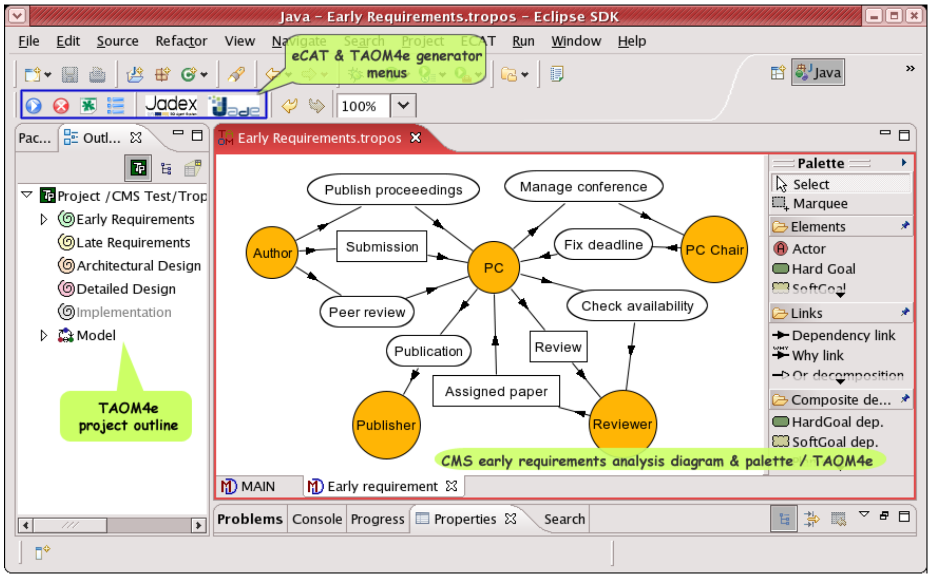


Fig. 3. A snapshot of the TAOM4e's GUI including: the tool's button menu (top); the project's artefacts browser (left); model views, e.g. the Early requirements Actor Diagram (centre); the modeller's palette (right).

In practice, using *TAOM4e*, the Early Requirements model is built by creating a first Actor Diagram into the project and adding actors, goals, etc. into the model using the graphical editor. Fig. 3 shows a view of the Early Requirements model (actor diagram) for the CMS case study. Circles represent actors, ovals the goals, rectangles the resources and the double arrows links between pairs of actors the dependencies between the two actors for the achievement of the goal or resource connected by the two dependency links. For every entity in the model, some properties, such as formal properties related to the Formal Tropos language [8], can be specified in the tool, according to the metamodel defined in [2].

AO modelling can be further pursued by decomposing a goal into sub-goals and by exploring the possible alternatives to achieve a goal. Alternatives are represented by OR-decomposition and characterized by multiple contributions. At this stage, also non-functional requirements can be represented as soft-goals. Choosing one alternative with respect to another, leads to different soft-goals achievement. By this way, it is possible to compare different alternatives and select the most appropriate one.

In Fig. 4, an Early Requirements goal diagram is shown. This diagram represents a (partial) view on the model. Only two actors of the model, PC and PC Chair, are represented with two goal dependencies, Manage conference and Decide deadlines. The goal Manage conference is analyzed from the point of view of its responsible actor, PC Chair, through an AND decomposition into sev-

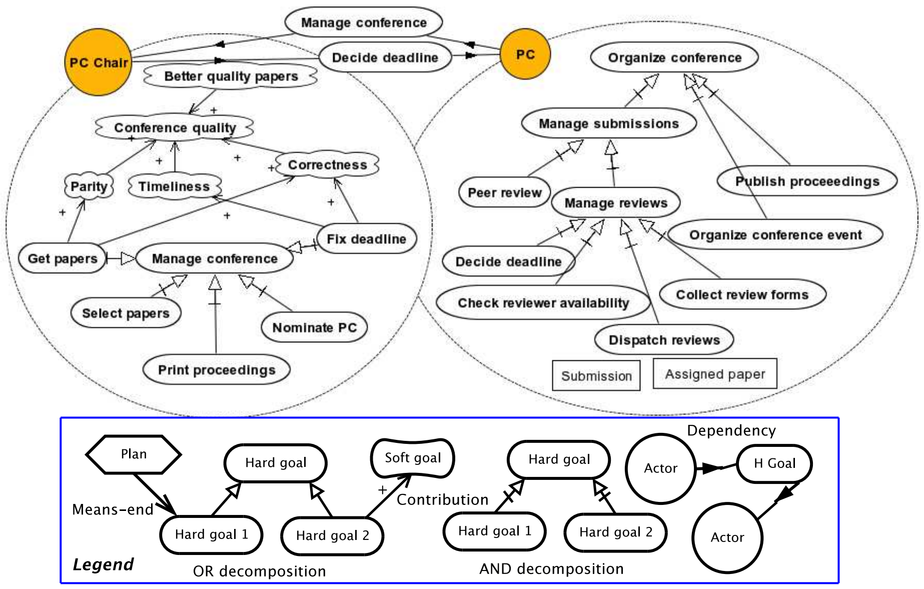


Fig. 4. Early Requirements of CMS: Goal Diagram.

eral goals: Get papers, Select papers, Print proceedings, Nominate PC and Decide deadlines. Moreover, softgoals can be specified inside the actor goal diagram, with their contribution relationships to/from other goals (see for example the softgoal Conference quality and the positive contribution relationship from the softgoal Better quality papers).

Goal diagrams can be dynamically created in *TAOM4e*. The tool allows, for every actor in the model, to open (close) their goal diagrams, which appear as balloons attached to the relative actors. This allows to dynamically visualize the internal perspective of each single actor. Notice also that the tool supports the analyst in identifying the elements to be analyzed. For instance, goals that have been delegated to an actor through dependency relationships, appears automatically in the actor goal diagrams, as for instance in the case of the PC Chair actor and the goal Manage conference in Fig. 4.

The results of the first phase are the Early Requirements model and the set of Actor and Goal diagrams produced during its specification.

The Late Requirements phase is intended to capture the changes in the domain caused by the introduction of the system-to-be and the actual properties of the system. The phase starts by introducing in the domain model a new actor representing the system-to-be.

A partial view of the resulting model is shown in Fig. 5 where the CMS System actor is represented. In practice, the analyst creates a new diagram inside the

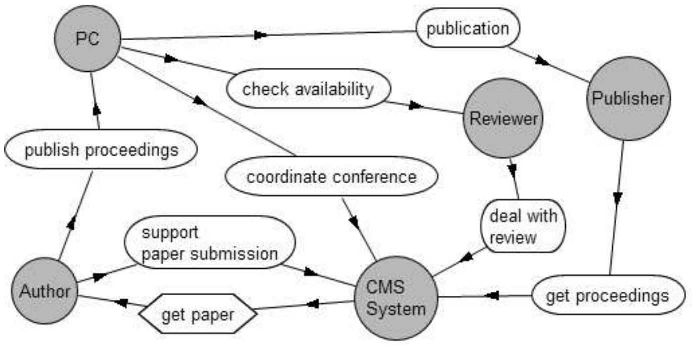


Fig. 5. Late Requirements: Actor Diagram.

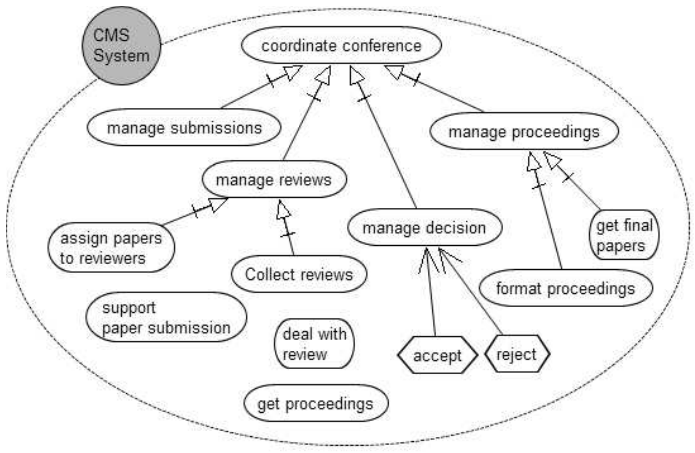


Fig. 6. Late Requirements: Goal Diagram.

project that, again, is a view on the model under construction, and adds the new actor. specifying its property of being a system actor.[8]

The driving analysis questions here can be stated as follows: *what are the goals that can be assigned to the system-to-be and which dependencies can be redirected from domain actors to the system?*

According to these questions, several existing or new dependencies can be respectively redirected and established between the other actors in the domain and the new CMS System actor, such as the new goal dependencies Coordinate conference and Manage proceedings.

These goals are then analyzed from the system actor perspective. In Fig. 6, the relative goal diagram is shown. The goals Coordinate conference and Man-

[8] The tool can be customized to show system actors with a different color with respect to domain actor to facilitate model reading.

age proceedings are decomposed in new sub-goals. Moreover, operative plans are specified and associated to the system goals as means to achieve them (means-ends relationships), such as in the case of the goal Manage decision that is operationalised by the plans accept and reject.

The resulting artefacts of this phase are the extended domain model and all the Late Requirements diagrams defined by the engineer. The model will be the input for the Design phases.

4 Design

The Late Requirements model is the basis for the definition of the actual system architecture. It is comprised by both the overall multi-agent system structure, and the detailed design for each single agent of the system.

The **Architectural Design** artefact consists of the system's overall structure: it is represented in terms of its sub-systems and of their inter-dependencies. Adopting the multi-agent system paradigm, sub-systems are agents that can act independently and communicate with others through message passing. In order to build the architectural design, the engineer will refine the system actor by introducing sub-actors, which are responsible for actually carrying out the system's top goals. The aim is to split the complexity of the system, which is described in terms of high-level goals, into smaller components, easier to design, to implement and to manage. During this refinement activity, the engineer has to face possible alternative decompositions. Among alternative decompositions, one that results in sub-systems with stronger internal cohesion and lower coupling should be selected.

TAOM4e gives the possibility to create an *Architectural Design* diagram for every system actor defined in *Late Requirements Analysis*. In this diagram, a dashed box associated to the system actor represents the system. In the box, new system agents can be created. Subsequently, a single goal, the whole goal tree or parts of them can be delegated from the system to the new system agents.

Fig. 7 displays the resulting architectural design diagram for the CMS System actor. Analyzing this actor's goal model (see fig. 6), the engineer should be able to extract a proper decomposition into sub-actors. In our example we introduce four new actors. The Conference Manager manages the top-level goal coordinate conference, delegated to the system by the program committee actor PC. The Paper Manager deals with the goal support paper submission from the domain actor Author, moreover some internal agents depend on it to manage papers. To do this, the agent depends on authors to get papers. Similarly, to the Review Manager and Proceedings Manager the corresponding goals are delegated.

Once the sub-actors have been modelled, together with the goals and tasks delegated to them, the next step consists in analyzing and detailing the goal model of these new agents. Similarly to the late requirements analysis phase, the engineer "opens" the balloon of an agent or creates a new view for the agent under consideration, to analyze the goals delegated to it. Goals can be decomposed and plans can be added as means for achieving goals.

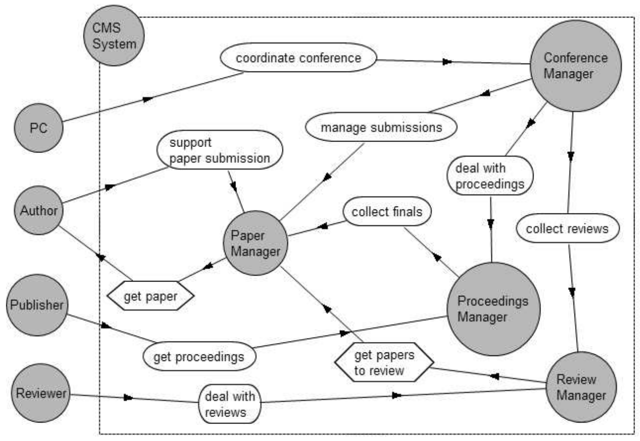


Fig. 7. Architectural Design: CMS System Decomposition into Sub-actors.

Fig. 8 shows an excerpt of the goal models for two of the sub-actors, namely Paper Manager and Proceedings Manager. We focus on the analysis of the goal get proceedings delegated from the Publisher actor, and the resulting dependency between the two system actors. The delegated goal is AND-decomposed into sub-goals, which are either operationalised by defining a plan or further decomposed. To be achieved, one of the sub-goals, deal with proceedings, causes the Proceedings Manager to depend on the Paper Manager for the goal collect finals.

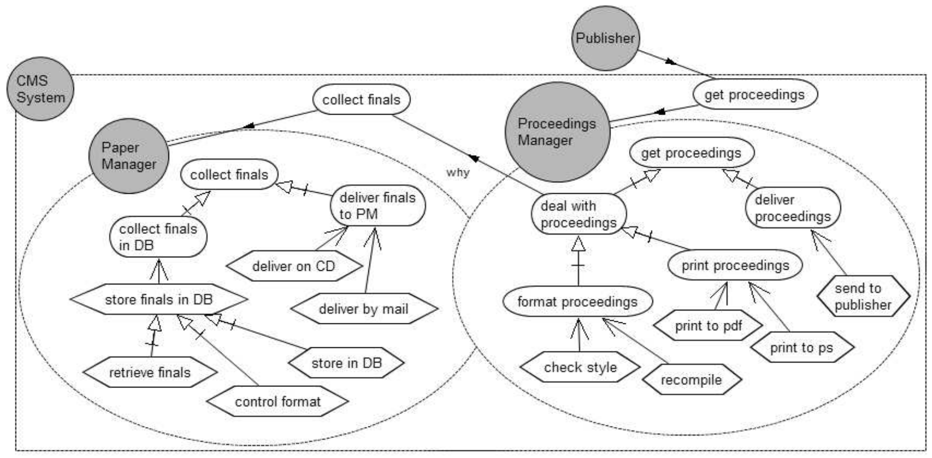


Fig. 8. Architectural Design: Simplified Goal Model of two Sub-actors of CMS.

Plans are defined as means to achieve the goals that are not delegated to other agents. Defining more than one plan for a goal (as for the goal format proceedings), leads to modelling alternatives. One possible way to format the proceedings is to recompile them, an alternative way is to control the style of posted papers. However, the applicability of the plans can depend on availability of resources (the source files in this example) and the selection of alternatives can be guided by looking at positive and negative contributions to softgoals, for example consistent formatting (not shown in the figure).

Opening the internal view of the PaperManager actor, the engineer can now find the goal collect finals that has been previously delegated to it. This goal can now be decomposed to sub-goals and operationalised by plans. Furthermore, plans can also be detailed, by decomposing them in AND and OR to more concrete sub-plans. See for instance the AND decomposition of the plan store finals in DB into the sub-plans retrieve finals, control format, store in DB, in Fig. 8.

The system design can be completed with the **Detailed Design** artefact that specifies in detail the plans associated to each agent goal and the agent interaction protocols.

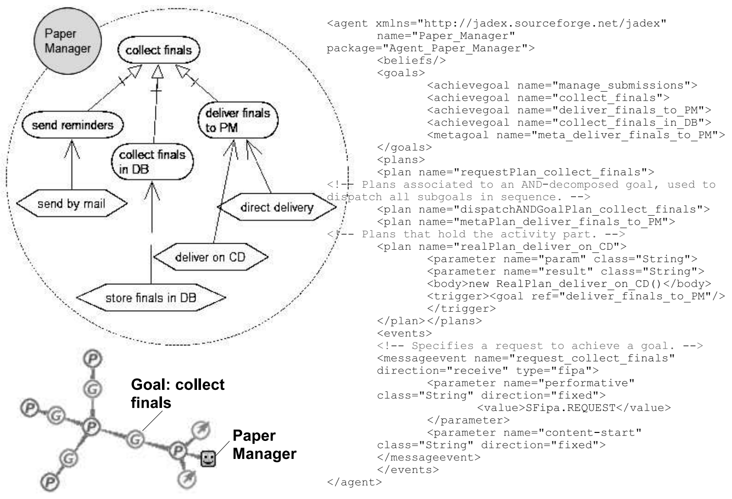
UML activity diagrams are automatically generated from the *Tropos* plan diagrams, by model transformation, using the *Tropos2UML* tool. The resulting diagrams can be further detailed and modified with any UML2 editor able to import files in XMI format. Sequence diagrams are associated to activities that contribute to the definition of the communication protocols used. Starting from these diagrams, *JADE* Behaviour code can be generated. These modelling steps are not used in the case study and therefore will be not further detailed in this paper, we focus instead on BDI code generation from goal models.

5 Code and Test Suites Generation

The goal models created in the design phase are the basis for the implementation of software agents. Using the *t2x* tool, *Jadex* agent definition files can be generated by selecting a system agent in the *GM* and starting the automatic generation process. Regarding the present case study, code was generated for the two system agents ProceedingsManager and PaperManager.

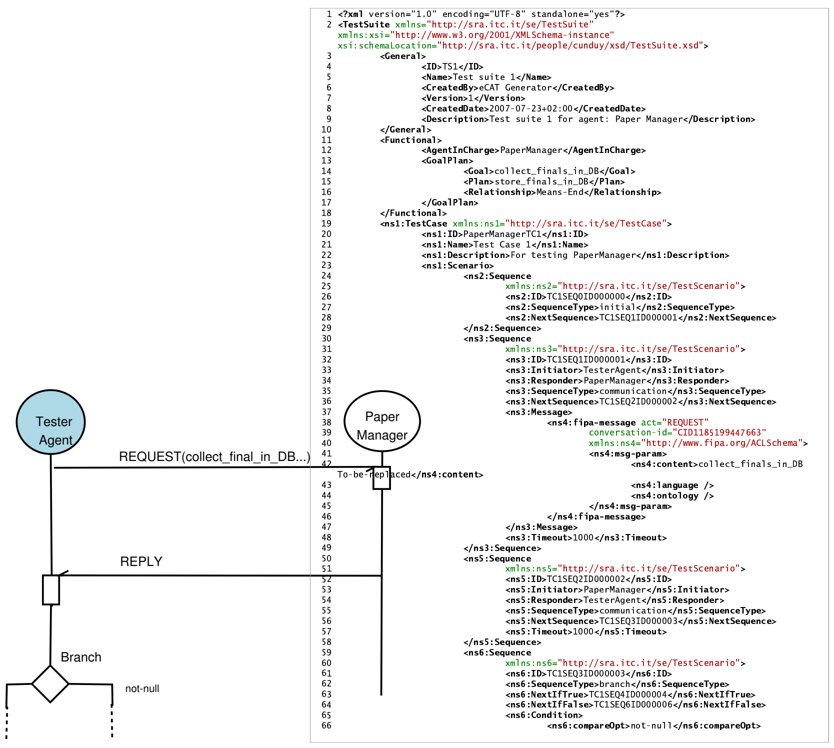
The generated code implements the agent's reasoning mechanisms needed to select correct plans at run-time to achieve desired goals. The *t2x* tool analyses a *GM* exploring goal decomposition trees. The goal hierarchy is mapped to *Jadex* goals along with Java files containing the decomposition logic, while plans are implemented in Java files and connected to the relative goals by a *triggering* mechanism. These goal decomposition graphs are also stored in the agent's belief base, together with all contributions to softgoals and dependencies to other agents. Therefore, at run-time the agent can control its behaviour by navigating the modelled goal graph.

The generated code skeleton can be executed on the *Jadex* platform. It exhibits a basic behaviour corresponding to the designed goal model and can be modified and customized as needed. In particular, it can be extended with code,



```xml
<agent xmlns="http://jadex.sourceforge.net/jadex"
       name="Paper_Manager"
package="Agent_Paper_Manager">
       <beliefs/>
       <goals>
              <achievegoal name="manage_submissions">
              <achievegoal name="collect_finals">
              <achievegoal name="deliver_finals_to_PM">
              <achievegoal name="collect_finals_in_DB">
              <metagoal name="meta_deliver_finals_to_PM">
       </goals>
       <plans>
       <plan name="requestPlan_collect_finals">
<!-- Plans associated to an AND-decomposed goal, used to
dispatch all subgoals in sequence. -->
       <plan name="dispatchANDGoalPlan_collect_finals">
       <plan name="metaPlan_deliver_finals_to_PM">
<!-- Plans that hold the activity part. -->
       <plan name="realPlan_deliver_on_CD">
              <parameter name="param" class="String">
              <parameter name="result" class="String">
              <body>new RealPlan_deliver_on_CD()</body>
              <trigger><goal ref="deliver_finals_to_PM"/>
              </trigger>
       </plan></plans>
       <events>
       <!-- Specifies a request to achieve a goal. -->
       <messageevent name="request_collect_finals"
       direction="receive" type="fipa">
              <parameter name="performative"
       class="String" direction="fixed">
                     <value>SFipa.REQUEST</value>
              </parameter>
              <parameter name="content-start"
       class="String" direction="fixed">
       </messageevent>
       </events>
</agent>
```

Fig. 9. Simplified goal diagram for PaperManager, part of generated *Jadex* XML code, and example *Jadex* run-time agent instance with activated goals and plans, visualized by the *Introspector* tool provided by the *Jadex* platform.



```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<TestSuite xmlns="http://sra.itc.it/se/TestSuite"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://sra.itc.it/people/cunduy/xsd/TestSuite.xsd">
        <General>
                <ID>TS1</ID>
                <Name>Test suite 1</Name>
                <CreatedBy>eCAT Generator</CreatedBy>
                <Version>1</Version>
                <CreatedDate>2007-07-23+02:00</CreatedDate>
                <Description>Test suite 1 for agent: Paper Manager</Description>
        </General>
        <Functional>
                <AgentInCharge>PaperManager</AgentInCharge>
                <GoalPlan>
                        <Goal>collect_finals_in_DB</Goal>
                        <Plan>store_finals_in_DB</Plan>
                        <Relationship>Means-End</Relationship>
                </GoalPlan>
        </Functional>
        <ns1:TestCase xmlns:ns1="http://sra.itc.it/se/TestCase">
                <ns1:ID>PaperManagerTC1</ns1:ID>
                <ns1:Name>Test Case 1</ns1:Name>
                <ns1:Description>For testing PaperManager</ns1:Description>
                <ns1:Scenario>
                        <ns2:Sequence
                                xmlns:ns2="http://sra.itc.it/se/TestScenario">
                                <ns2:ID>TC1SEQ0ID000000</ns2:ID>
                                <ns2:SequenceType>initial</ns2:SequenceType>
                                <ns2:NextSequence>TC1SEQ1ID000001</ns2:NextSequence>
                        </ns2:Sequence>
                        <ns3:Sequence
                                xmlns:ns3="http://sra.itc.it/se/TestScenario">
                                <ns3:ID>TC1SEQ1ID000001</ns3:ID>
                                <ns3:Initiator>TesterAgent</ns3:Initiator>
                                <ns3:Responder>PaperManager</ns3:Responder>
                                <ns3:SequenceType>communication</ns3:SequenceType>
                                <ns3:NextSequence>TC1SEQ2ID000002</ns3:NextSequence>
                                <ns3:Message>
                                        <ns4:fipa-message act="REQUEST"
                                                conversation-id="CID1185199447663"
                                                xmlns:ns4="http://www.fipa.org/ACLSchema">
                                                <ns4:msg-param>
                                                        <ns4:content>collect_finals_in_DB
  To-be-replaced</ns4:content>
                                                        <ns4:language />
                                                        <ns4:ontology />
                                                </ns4:msg-param>
                                        </ns4:fipa-message>
                                </ns3:Message>
                                <ns3:Timeout>1000</ns3:Timeout>
                        </ns3:Sequence>
                        <ns5:Sequence
                                xmlns:ns5="http://sra.itc.it/se/TestScenario">
                                <ns5:ID>TC1SEQ2ID000002</ns5:ID>
                                <ns5:Initiator>PaperManager</ns5:Initiator>
                                <ns5:Responder>TesterAgent</ns5:Responder>
                                <ns5:SequenceType>communication</ns5:SequenceType>
                                <ns5:NextSequence>TC1SEQ3ID000003</ns5:NextSequence>
                                <ns5:Timeout>1000</ns5:Timeout>
                        </ns5:Sequence>
                        <ns6:Sequence
                                xmlns:ns6="http://sra.itc.it/se/TestScenario">
                                <ns6:ID>TC1SEQ3ID000003</ns6:ID>
                                <ns6:SequenceType>branch</ns6:SequenceType>
                                <ns6:NextIfTrue>TC1SEQ4ID000004</ns6:NextIfTrue>
                                <ns6:NextIfFalse>TC1SEQ6ID000006</ns6:NextIfFalse>
                                <ns6:Condition>
                                        <ns6:compareOpt>not-null</ns6:compareOpt>
```

Fig. 10. Example of a Test Scenario. An excerpt of the XML specification is depicted in the right part.

generated by *UML2JADE*, corresponding to the activity diagrams that can be specified at detailed design.

As an example, Fig. 9 briefly shows the generated *Jadex* code, in XML format, of the agent Paper Manager. This fragment of code corresponds to the *Tropos* goal model on the top-left side of the figure, and its reasoning trace at run-time is presented on the bottom-left corner of the figure.

Following the goal-oriented testing methodology presented in [16], *eCAT* generates test suites for every elementary relationship, i.e. relationship between a goal and a plan. The underlining idea is to use the test suite as a guideline for the *Autonomous Tester Agent* to trigger the goal in order to verify the execution of the corresponding plan. In the case of CMS, *eCAT* takes the architectural diagram, Fig. 8, as an input and generate a set of test suites for each agent. Developers can choose when generating test suites which communication protocols the *Autonomous Tester Agent* will use to communicate with the agents of CMS. As an example, Fig. 10 illustrates a test suite that tests whether the agent PaperManager is able to fulfil the goal collect finals in DB or not. The graphical part of the figure gives an intuitive understanding of the test suite, formalized in XML: when executing test, the *Autonomous Tester Agent* will send a request that has "REQUEST" as its performative and the name of the goal collect finals in DB as message content to Paper Manager. Then, it will wait for a reply and decide to finish the test or to continue with other requests.

6 Discussion

For sake of simplicity we have not described iterations along different phases that usually occur in the development process. For instance, iterations along the Early and Late requirements phases, in order to explicit domain entities that are relevant when specifying the impact of the system-to-be in the original organizational setting, and that may have not been captured in the initial Early Requirements model. *Tropos* allows model refinement through iterative steps. This process is managed manually since, up to now, *TAOM4e* does not provide versioning functions. Moreover, formal techniques to support goal analysis and consistency checking of the requirements model have not been exploited.

We shall mention also the fact that the CMS case study offers interesting problems that have not been considered in this paper, due to lack of space. For instance, non-functional requirements, which may emerge in case of large-size conferences and may require more complex MAS architecture, should be taken into account.

Moreover, rules and norms that characterize the CMS domain were not modelled in the case study. For example, rules for manging possible conflicts between the reviewers and the authors of papers to be reviewed should have been modelled. We also have not addressed the rules related to the instances, such as those related to the number of reviews for every paper or the policy of distribution of the papers to the reviewers. Some of these rules can be represented in *Tropos* in the form of Linear Temporal Logics constraints imposed on the entities of the model via the Formal Tropos language. As pointed out in Section 3, *TAOM4e* allows for the representation of these constraints in the form of annotated properties on the model entities. Moreover, starting from the Early requirements phase, the formal annotations give the possibility to formally check the model via model checking techniques as described in [19]. An alternative approach to norm modelling with an AO approach is described in [22].

7 Conclusion and Future Work

This paper illustrated how to use the *Tropos* methodology and a set of supporting tools, to develop a MAS for the Conference Management System case study. In particular, analysis, design, code generation and testing activities have been illustrated together with examples of the resulting process's artefacts.

Work is ongoing to consolidate the tool-supported development process in *Tropos*. In particular, we are deserving particular effort to the integration of requirements and design modelling with the Goal Oriented testing methodology. Moreover, we are studying mechanisms for supporting automatic traceability between process artefacts, e.g. design artifacts and code.

References

1. F. Bellifemine, A. Poggi, and G. Rimassa. JADE: A FIPA Compliant agent framework. In *Practical Applications of Intelligent Agents and Multi-Agents*, pages 97–108, April, 1999.
2. D. Bertolini, A. Novikau, A. Susi, and A. Perini. TAOM4E: an Eclipse ready tool for Agent-Oriented Modeling. Issue on the development process. Technical report, Fondazione Bruno Kessler - irst, 2006.
3. 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.
4. V. Bryl, P. Giorgini, and J. Mylopoulos. Designing cooperative IS: Exploring and evaluating alternatives. In *OTM Conferences (1)*, pages 533–550, 2006.
5. K. Dam and M. Winikoff. Comparing Agent-Oriented Methodologies. In *Proceedings of the 5th Int'l Bi-Conference Workshop on AgentOriented Information Systems (AOIS), Melbourne, Australia*, 2003.
6. S. A. DeLoach. Modeling organizational rules in the multi-agent systems engineering methodology. In R. Cohen and B. Spencer, editors, *Canadian Conference on AI*, volume 2338 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002.
7. M.-P. G. Federico Bergenti and F. Zambonelli, editors. *Methodologies and Software Engineering for Agent Systems : The Agent-Oriented Software Engineering Handbook*. Springer, 2004.
8. A. Fuxman, L. Liu, J. Mylopoulos, M. Roveri, and P. Traverso. Specifying and analyzing early requirements in tropos. *Requir. Eng.*, 9(2):132–150, 2004.
9. 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), Aug. 2001. IEEE Computer Society.

10. P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Modeling security requirements through ownership, permission and delegation. In *Proceedings of the 13th IEEE International Requirements Engineering Conference (RE'05)*, 2005.
11. P. Giorgini, J. Mylopoulos, A. Perini, and A. Susi. The Tropos Methodology and Software Development Environment. In P. Giorgini, N. Maiden, J. Mylopoulos, and E. Yu, editors, *Social Modelling for Requirements Engineering.* MIT Press. To appear.
12. P. Giorgini, J. Mylopoulous, and R. Sebastiani. Goal-Oriented Requirements Analysis and Reasoning in the Tropos Methodology. *Engineering Applications of Artificial Intelligence*, 18(2):159–171, 2005.
13. B. Henderson-Sellers and P. Giorgini, editors. *Agent-Oriented Methodologies.* Idea Group Inc., 2005.
14. M. Morandini. Knowledge Level Engineering of BDI Agents. Master's thesis, Dept. of Computer Science, University of Trento, Italy, 2006. Available at `http://dit.unitn.it/~morandini/resources/ThesisMirkoMorandini.pdf`.
15. M. Morandini, L. Penserini, A. Perini, and A. Susi. Refining goal models by evaluating system behaviour. In *8th International Workshop on Agent-Oriented Software Engineering, AAMAS*, May 2007.
16. D. C. Nguyen, A. Perini, and P. Tonella. A goal-oriented software testing methodology. In *8th International Workshop on Agent-Oriented Software Engineering, AAMAS*, May 2007.
17. 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.
18. L. Penserini, A. Perini, A. Susi, and J. Mylopoulos. From Stakeholder Intentions to Agent Capabilities. In *Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'07)*, Haway, USA, 2007. ACM Press.
19. A. Perini and A. Susi. Agent-Oriented Visual Modeling and Model Validation for Engineering Distributed Systems. *Computer Systems Science & Engineering*, 20(4):319–329, 2005.
20. A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: Implementing a bdi-infrastructure for jade agents. *EXP - in search of innovation (Special Issue on JADE)*, 3(3):76–85, 9 2003.
21. 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.
22. A. Siena. Engineering Normative Requirements. In *Proceedings of the First International Conference on Research Challenges in Information Science, RCIS 2007, Ouarzazate, Morocco*, pages 439–444, 2007.
23. R. G.-S. Souza and A. Perini. Analyzing requirements of knowledge management systems with the support of agent organizations. *Journal of the Brazilian Computer Society (JCBS)*, 11(1):51–62, 2005. ISSN 0104-6500.
24. E. Yu. *Modelling Strategic Relationships for Process Reengineering.* PhD thesis, University of Toronto, Department of Computer Science, University of Toronto, 1995.