The Tropos Software Engineering Methodology

Mirko Morandini, Fabiano Dalpiaz, Cu Duy Nguyen, and Alberto Siena

Abstract The agent-oriented software engineering methodology *Tropos* offers a structured development process for the development of socio-technical systems. Such systems explicitly recognise the interplay between social actors (humans and organisations) and technical systems (software). *Tropos* adopts the state-of-the-art *i** requirements modelling language throughout the development cycle, giving special attention to the early phases of domain and requirements analysis. The system is modelled in terms of the goals of the involved actors and their social interdependencies, allowing for a seamless transition from the requirements to the design and potentially to an an agent-oriented implementation. *Tropos* prescribes a limited set of domain-independent models, activities, and artefacts, which can be complement with domain- and application-specific ones.

Acknowledgements The authors thank Angelo Susi (Fondazione Bruno Kessler), Anna Perini (Fondazione Bruno Kessler), and Paolo Giorgini (University of Trento) for their support.

M. Morandini, C. D. Nguyen
Fondazione Bruno Kessler, via Sommarive 18, 38123 Trento, Italy
e-mail: {morandini|cunduy}@fbk.eu

F. Dalpiaz
Department of Computer Science, University of Toronto, 40 St. George Street, M5S 2E4, Toronto, Ontario, Canada
e-mail: dalpiaz@cs.toronto.edu

A. Siena
DISI, Università di Trento, via Sommarive 5, 38123 Trento, Italy
e-mail: siena@disi.unitn.it

1 Introduction

Tropos is a comprehensive, agent-oriented methodology for developing socio-technical systems. Such systems explicitly recognise the existence of and interplay between technical systems (software) and social actors (humans and organisations). *Tropos* adopts a requirement-driven approach to software development, recognizing a pivotal role to the modelling of domain stakeholders (social actors) and to the analysis of their goals and interdependencies, before designing a technical system-to-be that supports the social actors. System design results in the specification of a multi-agent system. The methodology was first introduced in [4] and has been extended in different ways in the last decade. For instance, its modelling language has been adapted to support the analysis of crucial issues in distributed systems, such as trust, security and risks [10, 1].

Tropos encompasses all the software development phases, from *Early Requirements* to *Implementation and Testing*. *Tropos* introduces an *Early Requirements Analysis* phase in software development in which the analysts consider the stakeholders, their strategic goals, and the organisational aspects of the system *as it is*, before the software system under design comes into play.

Throughout the design phases, the aim of *Tropos* is to understand and analyse the goals of the stakeholders and to operationalise them, obtaining the requirements for the software system to be built. *Tropos* was proposed to ease traceability issues in the software development life-cycle, by relying upon consistent concepts and artefacts throughout the development phases. As a result, problems that arise during software construction can be traced back to their original requirements, and an advanced analysis can be conducted to check whether a specific implementation satisfies a stakeholder's goals. Besides this, *Tropos* also provides guidelines for implementation and testing. Designed artefacts are used to generate agent-oriented prototypes and for the derivation of test cases, whose execution result can tell the fulfilment of stakeholders' goals (see [18]).

Several support tools for *Tropos* are available: *TAOM4e* covers the entire development cycle [21, 13], *t2x* supports generating agent-oriented implementations [14], the T-Tool [8] provides model checking for *Tropos* specifications, the GR-Tool supports formal reasoning on goal models [9], multi-agent planning enables the selection among alternative networks of delegations [6].

The *Tropos* methodology was applied to various research and industrial case studies. Research contributions, including several Ph.D. theses from various universities and research centers including, among others, FBK Trento and the Universities of Trento, Toronto, Louvain, Haifa, and Recife, consolidated and extended the *Tropos* methodology in several directions, covering topics such as security, norms, risks, agent testing, adaptive systems, socio-technical systems, traceability, services, formal analysis, and model interchange. Furthermore, various empirical studies conducted on *Tropos*, its extensions and concurrent methodologies, demonstrated its applicability in different domains [11, 16, 5].

This chapter provides guidance on how to use *Tropos* to develop a Multi-Agent System (MAS), performing analysis and design activities, generating code and per-

forming testing on it, with the support of a set of tools. Moreover, it enables 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. Following the IEEE FIPA standard XC00097A for agent-oriented software engineering, first the methodology life cycle and meta-model are presented. Then, each phase of the methodology is presented by describing participating roles, included activities and resulting work products. Finally, the dependencies between the work products are outlined.

2 The Tropos Process Lifecycle

The software development process in *Tropos* consists of five subsequent phases, shown in Fig. 1:

- *Early Requirements Analysis* is concerned with understanding and modelling the existing organizational setting where the system-to-be will be introduced (the so-called "as-is" setting). The organization is represented in terms of goal-oriented actors that socially depend one on another to fulfil their respective goals.
- *Late Requirements Analysis* starts from the output of the early requirements, and introduces the system-to-be in the organizational setting.
- *Architectural Design* defines the system-to-be's overall architecture in terms of interacting subsystems (agents). These agents are those to be implemented.
- *Detailed Design* further refines the system specification. It defines the functionalities to be implemented in each agent as well as the interaction protocols.
- *Implementation and Testing* are concerned with the actual development of the system agents and verifying whether they operate and interact as specified, respectively.

Even though not expressly shown in the figure, backwards iterations are possible and often needed, since the analysis carried out in a phase can provide feedback for the refinement of a previous phase. For example, while modelling the system-to-be in the late requirements phase, additional actors in the organizational setting may be identified. Their analysis is carried out through another iteration of the early requirements phase.

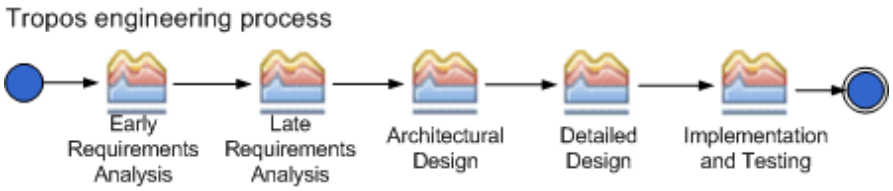


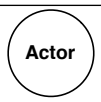
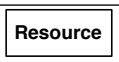
Fig. 1 The phases of the Tropos software engineering process

The *Tropos* software engineering process is model-based. Diagrammatic requirements and design models are used and refined throughout the process. These models are created using a variant of the conceptual modelling language i* [24]. Modelling activities are central to the first four phases in the software development process, from early requirements analysis to detailed design. The basic concepts of the goal-oriented modelling language (see Section 3) are those of actor, goal, plan and social dependency. Goal modelling is complemented by UML activity and sequence diagrams, in the detailed design phase.

Goal modelling can be performed using *i** modelling tools such as *TAOM4e* [13], which implements the *Tropos* metamodel presented in Section 3, provides explicit support for the different modelling phases, and is able to derive skeletons of a BDI-based agent implementation from goal models.

3 The Tropos Metamodel and Language

The following table describes the key concepts in the *Tropos* language and their graphical notation. The definition of the concepts is summarized from [4] and [20].

Actor	It models an entity that has strategic goals and intentionality within the system or the organizational setting. It represents a physical, social or software agent as well as a role or position.	
Goal	It represents actors' strategic interests. We distinguish *hard goals* (often simply called *goals*) from *softgoals*. The latter have no clear-cut definition and/or criteria for deciding whether they are satisfied or not, and are typically used to describe preferences and quality-of-service demands.	
Plan	It represents, at an abstract level, a way of doing something. The execution of plan can be a means for satisfying a goal or for *satisficing* (i.e. sufficiently satisfying) a softgoal.	
Resource	It represents a physical or an informational entity.	
Dependency	It is specified between two actors to indicate that one actor depends, for some reason, on the other in order to attain some goal, execute some plan, or deliver a resource.	

Capability	It represents both the *ability* of an actor to perform some action and the *opportunity* of doing this. These concepts are represented by a plan, together with the link to the goal which is the means to execute the plan, and eventual positive or negative contribution links to softgoals, which describe the opportunity of executing this plan in favour of alternative ones.

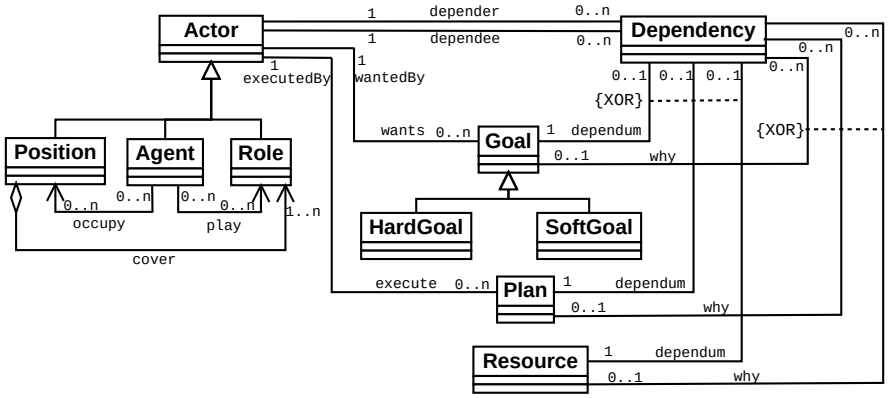


Fig. 2 The UML class diagram specifying the strategic part of the *Tropos* metamodel.

The *Tropos* metamodel has been specified in [23]. Here we recall the concepts and relationships of the language defined in the metamodel.

Concerning the concepts related to the *Tropos* actor diagram (as shown in Figure 2), a "position" can *cover* $1 \ldots n$ roles, whereas an "agent" can *play* $0 \ldots n$ "roles" and can *occupy* $0 \ldots n$ "positions". An "actor" can have $0 \ldots n$ "goals", which can be both hard and soft-goals and are wanted by 1 actor.

An actor *dependency* is a quaternary relationship relating two actors, the depender and the dependee, by a dependum, which can be a goal, a plan, or a resource and describes the nature of the relationships between the two actors. The *why* relationship between the dependum and another concept (goal, plan or resource) in the scope of the depender actor allows to specify the reason for the dependency.

Figure 3 shows the concepts of the *Tropos* goal diagram. The distinctive concept of goal is represented by the class *Goal*. Goals can be analysed, from the point of view of an actor, by *Boolean decomposition*, *Contribution analysis* and *Means-end analysis*. *Decomposition* is a ternary relationship which defines a generic boolean decomposition of a root goal into subgoals, that can be an *AND*- or an *OR-decomposition* specified via the attribute Type in the class *Boolean Decomposition* specialization of the class *Decomposition*. *Contribution Analysis* is a ternary relationship between an *actor*, whose point of view is represented, and two goals. Contribution analysis strives to identify goals that can contribute positively or negatively towards the fulfilment of other goals (see association relationship labelled *contribute*

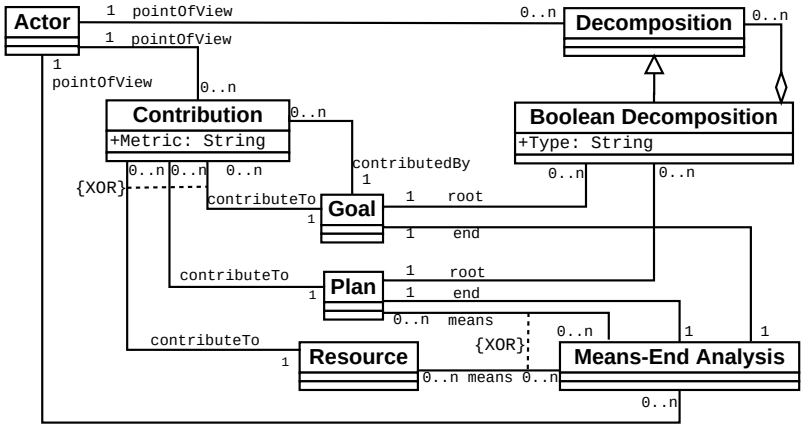


Fig. 3 The UML class diagram specifying the concepts related to the goal diagram in the *Tropos* metamodel.

in Figure 3). A contribution can be annotated with a qualitative metric, as proposed in [7], denoted by $+, ++, -, --$. In particular, if the goal g_1 contributes positively to the goal g_2, with metric $++$ then if g_1 is satisfied, so is g_2. Analogously, if the plan p contributes positively to the goal g, with metric $++$, this says that p fulfils g. A $+$ label for a goal or plan contribution represents a partial, positive contribution to the goal being analysed. With labels $--$, and $-$ we have the dual situation representing a sufficient or partial negative contribution towards the fulfilment of a goal. The *Means-end relationship* is also a ternary relationship defined among an *Actor*, whose point of view is represented in the means-end analysis, a goal (the end), and a *Plan*, *Resource* or *Goal*, representing the means which will be able to satisfy that goal, i.e. the *operationalisation* of the goal.

Plan analysis in *Tropos* is specified in Figure 3. *Means-end analysis* and *AND/OR decomposition*, defined above for goals, can also be applied to plans. In particular, AND/OR decomposition allows for modelling the detailed plan structure.

4 Early Requirements Phase

The Early Requirements (ER) Phase concerns the understanding of the organizational context within which the system-to-be will eventually function. During early requirements analysis, the requirements engineer identifies the domain stakeholders and acquires through them domain knowledge about the organizational setting. The organizational setting is further detailed by eliciting and elaborating the needs, preferences and responsibilities of the stakeholders. This domain knowledge is captured in ER models describing social actors, who have goals and depend on each other for goals to be fulfilled, plans to be performed, and resources to be furnished.

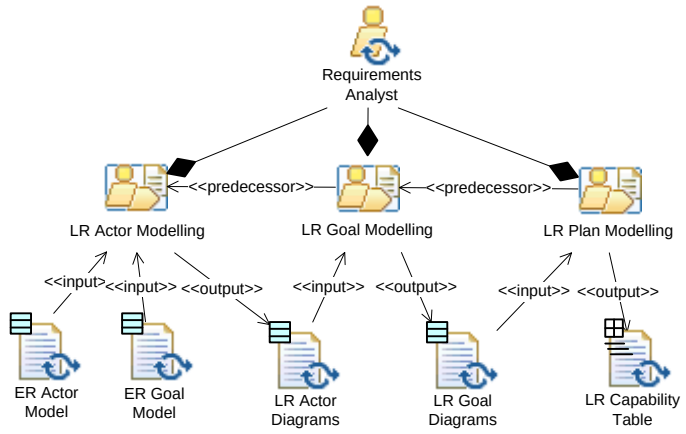


Fig. 4 The structure of the Early Requirements phase described in terms of activities, involved roles and work products.

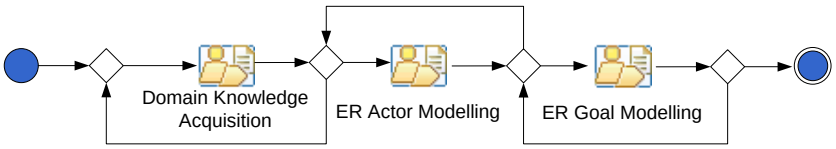


Fig. 5 Flows of activities in the Early Requirements phase.

4.1 Process Roles

Two Roles are involved in this phase: Requirements Analysts and Domain Stakeholders.

4.1.1 Requirements Analyst

Requirements Analysts gather knowledge about the domain and transform them into models of early requirements. They are in charge of:

- Analysing domain documents
- Setting up interviews and focus groups with Domain Stakeholders
- Preparing scenarios
- Gathering information elicited during interviews and focus groups
- Producing ER Actor models
- Producing ER Goal models

4.1.2 Domain Stakeholder

Domain stakeholders own the knowledge about the domain. They are in charge of:

- Providing domain knowledge to Requirements Analysts
- Participating to interviews and focus groups
- Informally validating the requirements models

4.2 Activity Details

4.2.1 Domain knowledge acquisition

Domain Knowledge acquisition consists in the activity of gathering knowledge from domain stakeholders. It is performed by analysing documents, stakeholder knowledge and production of scenarios, and acquired by setting up stakeholder interviews or focus groups. The output of this activity consists of domain documentation that will be the basis for the next two activities in this phase.

In the Conference Management System scenario, during this phase the domain is investigated to capture its fundamental organizational setting, such as domain actors and their responsibilities. 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 answers to these question contain information about stakeholders such as the Program Committee, the Program Chair, the paper submission and peer reviewing mechanism and so on. They allow for example to insert the Publisher as a major stakeholder in the domain, and demote the Vice Chair as not relevant with respect to the CMS mechanism.

4.2.2 ER actor modelling

During Early Requirements actor modelling activity, stakeholders, their desires, needs and preferences are modelled in *Tropos* in terms of actors, goals, and actor dependencies. 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 two actors. The activity produces as output an ER actor model using the delegation step in the design process.

The Conference Management System domain is modelled in terms of its main stakeholders (actors), as shown in Fig. 6: they are papers' authors, modelled as the actor Author; the conference's program committee and its chair – the PC PC Chair actors respectively –, papers reviewers, modelled as the actor Reviewer and the proceedings publisher, actor Publisher. Dependency relationships between actors are identified, 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 softgoals and resources, according to the *Tropos* modelling process.

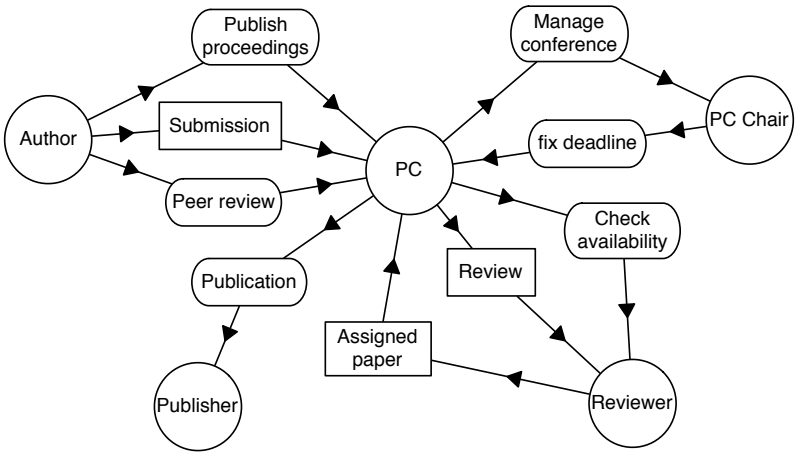


Fig. 6 The Early requirements Actor Diagram for the Conference Management System domain.

4.2.3 ER goal modelling

The Early Requirements goal modelling activity is intended to model the goals of each given actor, producing a goal model that represent the actor's rationale. This activity includes a modelling algorithm comprised by three steps:

- Refine. Goals are refined from a higher abstraction level (typically, the top level strategic goals coming form the actor model) into more fine-grained goals. The refinement can include an And- (i.e. decomposition) and Or- (i.e. alternative) relations among the refining goals.
- Delegate. Goals that are desired by an actor but are able to be satisfied by another actor are delegated from the former to the latter.
- Contribute. Contribution relations are established between goals whenever the achievement of a certain goal helps in the achievement of another goal or softgoal.

Additional stakeholder needs and preferences can be modelled in this activity by means of adding new goals and softgoals, and dependency links representing social relationships. Early requirements analysis consists of several refinements of the models identified so far, and e.g., further dependencies can be added through a means-ends analysis of each goal.

In Fig. 7, 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 Fix deadlines. The goal Manage conference is analysed from the point of view of its responsible actor, PC Chair, through an And- decomposition into several 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).

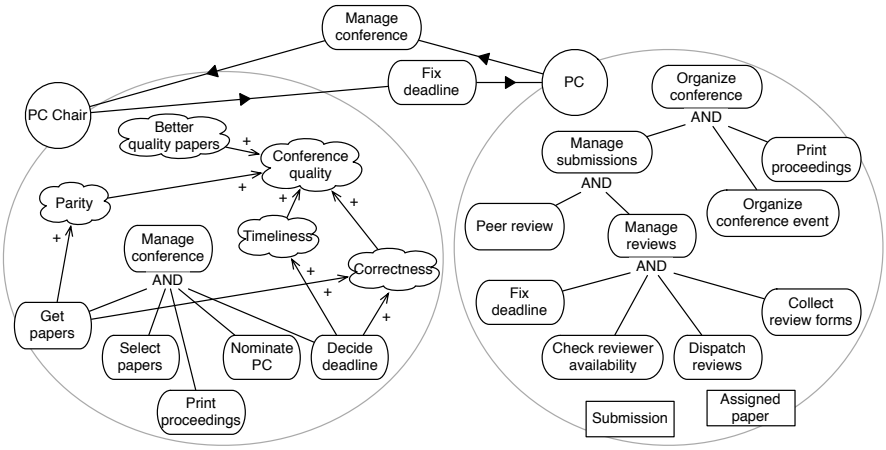


Fig. 7 Early Requirements of CMS: Goal Diagram.

4.3 Work Products

Table 1 shows the work products for the Early Requirements phase. The relationships of these work products with the *Tropos* meta-model elements are described in the following Figure 8. Since these relationships are valid for all goal modes (including goal and actor diagrams and the capability table) throughout the *Tropos* process, the figure does not attribute the work products to a specific phase.

4.3.1 Work Products Examples

The ER Actor Diagram is illustrated by Fig. 6, which shows the actors in the Early Requirements phase as long as their social dependencies. Fig. 7 illustrates an ER Goal Diagram for the System-to-be actor.

Table 1 Work products for the Early Requirements phase

Name	Description	Work Product Kind
Domain Documentation	It contains the knowledge about the domain as gathered by requirements analysts.	Composite
ER Actor Diagram	It represents the actors identified in the domain and the strategic dependencies among them.	Structural
ER Goal Diagram	For each identified actor, it represents the gathered hard and soft goals that form the rationale of that actor, decomposed into sub-goals and operationalised through plans.	Structural

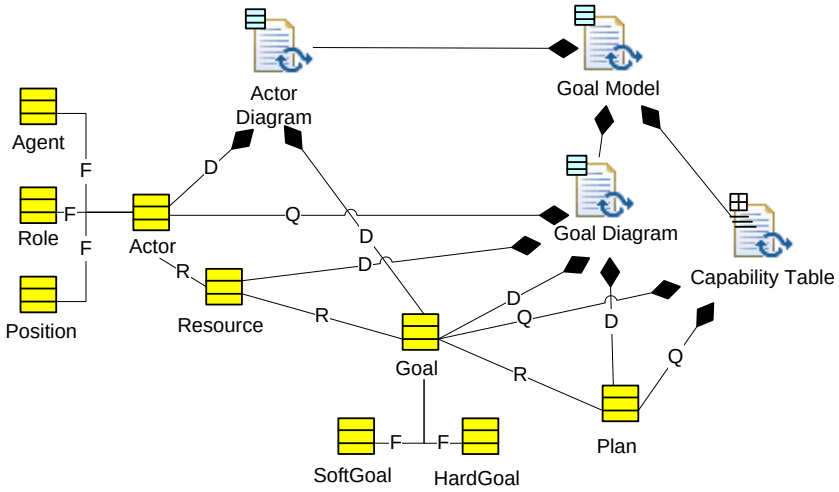


Fig. 8 General structure of a *Goal Model*, composed of actor- and goal diagrams and the capability table, in relation to the meta-model elements, as used in the early and late requirements analysis phases and in the architectural and detailed design phases (D: define, R: relate, F: refine, Q: query).

5 Late Requirements Phase

The Late Requirements phase is concerned with the definition of functional and non-functional requirements of the system-to-be. This is accomplished by treating the system as another actor (or a small number of actors representing system components) who are dependers or dependees in dependencies that relate them to external actors. The shift from early to late requirements occurs when the system actor is introduced and it participates in delegations from/to other actors. Here, modelling activities consist of introducing the system-to-be as a new actor with specific dependencies from/to stakeholder actors. These dependencies lead to the identification of system goals, that will be further analysed in terms of sub-goals, of plans providing

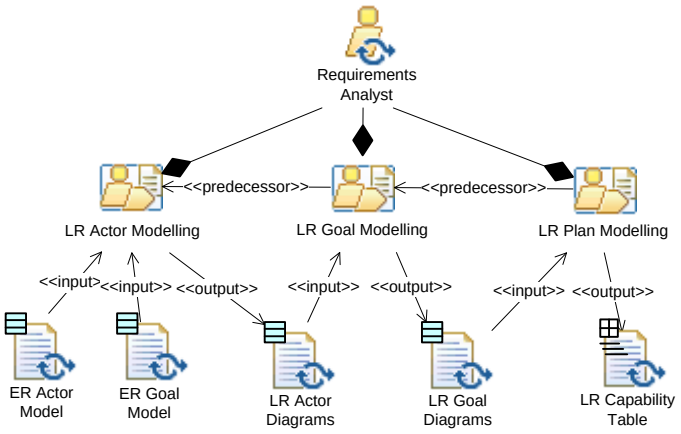


Fig. 9 The structure of the Late Requirements phase described in terms of activities, involved roles and work products.

means for their achievement, of positive and negative contributions to stakeholders' preferences (typically modelled as softgoals).

5.1 Process Roles

The Late Requirements Phase is accomplished by Requirements Analysts.

5.1.1 Requirements Analyst

During this phase, Requirements Analysts are in charge of:

- Introducing the system-to-be as a new actor
- Delegating stakeholder goals to the system to be through dependency links
- Refining the top goals assigned to the system to be to leaf-level goals
- Identifying the capabilities of the system-to-be and model them as plans

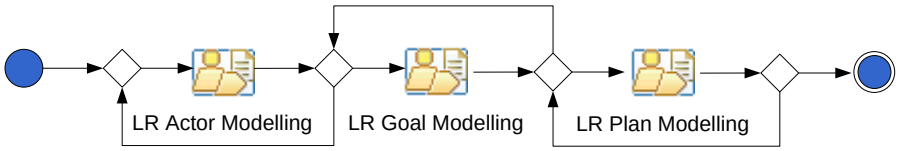


Fig. 10 The flow of activities in the Late Requirements phase.

5.2 Activity Details

5.2.1 LR actor modelling

During Late Requirements actor modelling, the System-to-be is introduced into the models as a new actor. Stakeholder needs and preferences are assigned to the System-to-be actor by establishing goal dependency links from stakeholder actors to the System actor. Top-level goals received by stakeholders form the functionalities which the system is responsible for.

A partial view of the LR actor model for the CMS domain is shown in Fig. 11 where the CMS System actor is represented.

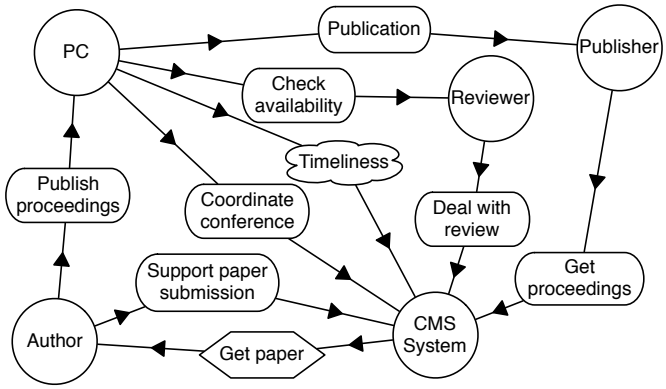


Fig. 11 Late Requirements: Actor Diagram.

5.2.2 LR goal modelling

During Late Requirements goal modelling the top-level goals of the system-to-be are refined into more fine-grained goals, resulting in system-specific goal models. This activity includes the same goal modelling algorithm used in ER goal modelling.

The System's goals are And- or Or- decomposed into more fine-grained goals. These goals are analysed *from the system actor perspective*.

In Fig. 12, the relative goal diagram is shown for the CMS domain. The goals Coordinate conference and Manage 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.

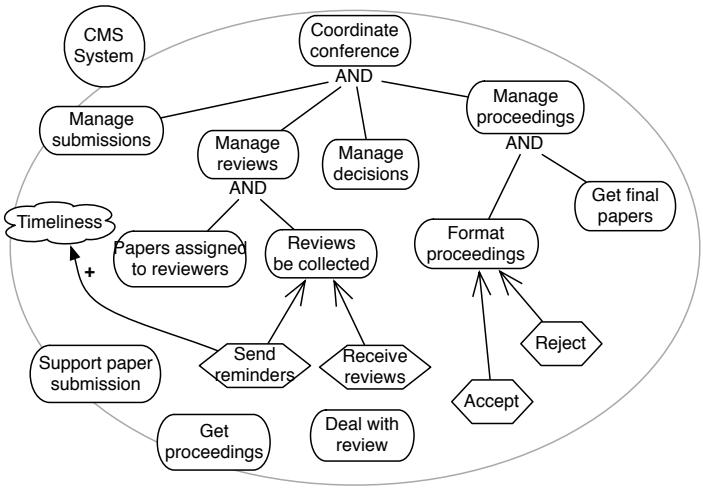


Fig. 12 Late Requirements: Goal Diagram.

5.2.3 LR plan modelling

During plan modelling actor goals are operationalised into concrete plans. If an actor owns the capability to fulfil a leaf-level goal by performing a specific plan, this is modelled into the goal model as a plan and bound through a means-end relation to the goal. Otherwise, the goal may require to be delegated to some other actor. This activity produces as output a modified version of the goal model and a a capability table. The capability table which consists in a view on the goal model, which highlights the capabilities identified for the actors.

In Fig. 12, the two plans Accept and Reject operationalise the goal Manage decision.

5.3 Work Products

Table 2 shows the work products for the Late Requirements phase. For the content of these work products refer to the general Figure 8.

Table 2 Work products for the Late Requirements phase.

Name	Description	Work Product Kind
LR Actor Diagram	It represents the System-to-be as a new actor of the model, and the functionalities assigned to it as functional dependencies from domain actor to the system actor.	Structural
LR Goal Diagram	It represents the operationalisation of the System-to-be strategic top goals in terms of tactical goals, including alternative ways to achieve them.	Structural
LR Capability Model	It represents the operationalisation of the System-to-be functional goals in terms of plans and contribution to softgoals.	Structured

5.3.1 Work Products Examples

The LR Actor Diagram is illustrated by Fig. 11, which shows the actors in the Late Requirements phase as long as their social dependencies. Fig. 12 illustrates an LR Goal Diagram for the System-to-be actor. Table 3 illustrates the LR Capability Model.

Table 3 LR Capability Model example.

Goal	Capabilities
Format proceedings	Accept; Reject
Reviews be collected	Send reminders, Receive reviews
...	...

6 The Architectural Design Phase

The *Architectural Design* (AD) phases drives the definition of the actual system architecture. It comprises both the overall multi-agent system structure and the detailed design for each single agent of the system. The phase is outlined in Fig. 13 (activity flow) and Fig. 14 (structural view), while details are provided in the following sub-sections.

The produced artefact consists of the system's overall structure, which is represented in terms of its sub-systems (*AD Goal Diagrams*), their inter-dependencies

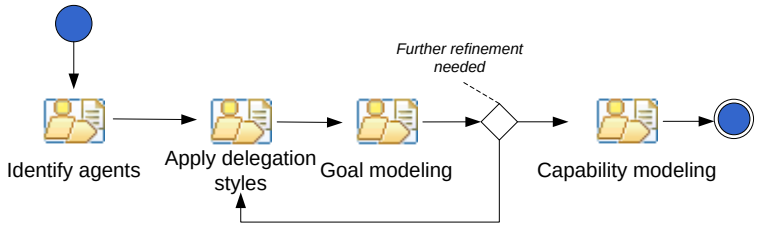


Fig. 13 Flow of activities in the *Tropos* Architectural Design phase.

(*AD Actor Diagram*), and their capabilities (*AD Capability Table*). Adopting the multi-agent system paradigm, sub-systems are autonomous agents that communicate through message passing.

6.1 Process roles

Two roles are involved: the *System Architect* and the *Agent Designer*.

6.1.1 System Architect

It is responsible for refining the system actor (*LR Goal Diagram*) by introducing system agents, which are delegated responsibility for the fulfilment of the system's

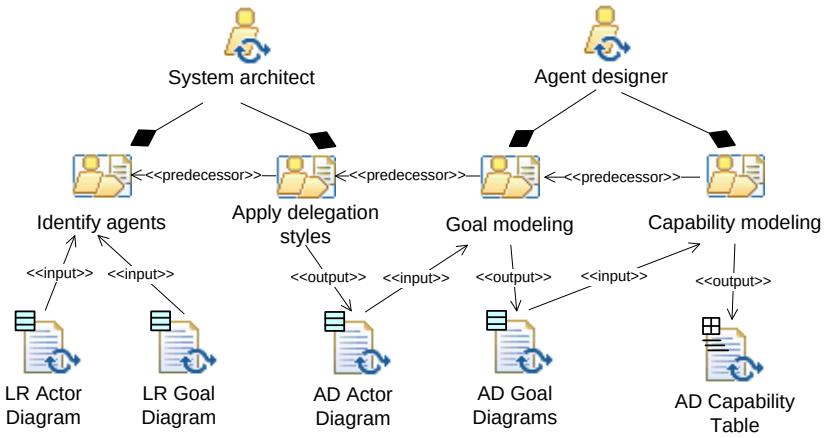


Fig. 14 The structure of the Architectural Design phase described in terms of activities, involved roles and work products.

goals. While delegating responsibilities, the architect may apply organizational patterns [12] so as to structure the relationships between and responsibilities of agents.

6.1.2 Agent Designer

Given the *AD Actor Diagram* created by the *System Architect*, it is responsible for detailing the identified agents by applying goal modelling (as in the early and late requirements phases). Additionally, the agent designer is responsible for performing capability modelling, which defines how each agent will fulfil the goals it is responsible for.

6.2 Activity Details

We detail each of the activities of the phase, showing the main involved artefacts, as well as examples from the CMS case study.

6.2.1 Identify agents

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. These components are autonomous agents.

6.2.2 Apply delegation styles

The identified agents take up responsibilities (through delegations) from the system actor. In other words, the goals of the system actor (*LR Goal Diagram*) are delegated to specific agents. During this activity, when applicable, organizational patterns [12] (e.g., structure-in-five or joint-venture) can be applied to decide how to relate the agents. Fig. 15 displays the resulting architectural design diagram for the CMS System actor. Analysing this actor's goal model (see Fig. 12), the engineer should be able to extract a proper decomposition into agents.

In our example we introduce four new agents. The Conference Manager manages the top-level goal Coordinate conference, delegated to the system by the program committee actor PC. The Paper Manager gets goal Support paper submission delegated from the domain actor Author. Further, some internal agents depend on it to manage submissions. To do this, the agent depends on authors to get papers. Similarly, the Review Manager and Proceedings Manager get goals delegated from the Reviewer and the Publisher, respectively.

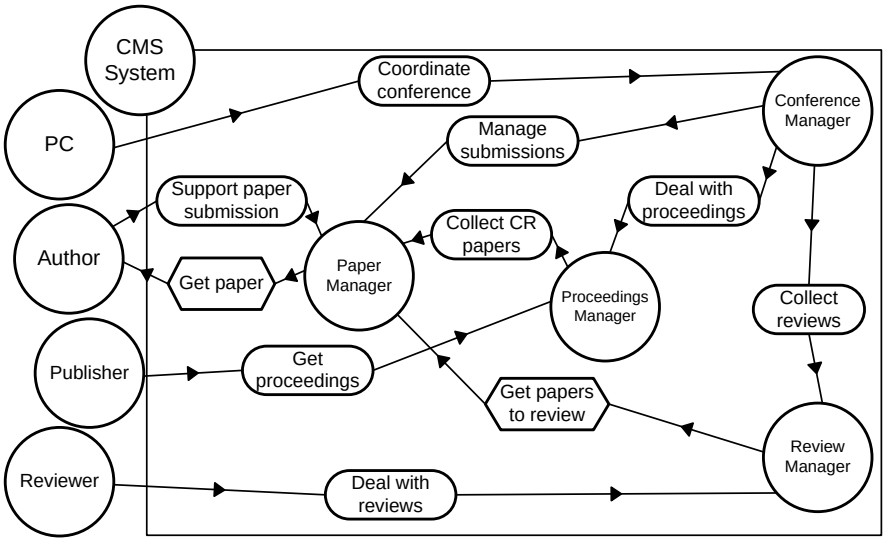


Fig. 15 Architectural Design: CMS System Decomposition into agents.

6.2.3 Goal modelling

Once the sub-actors have been modelled and have been assigned responsibilities through goal and task delegation, each of them is analysed by building a goal model. For more details about this activity, look at the goal modelling activity in the early and late requirements phases.

Fig. 16 shows an excerpt of the goal models for two agents, namely Paper Manager and Proceedings Manager. We focus on the analysis of the goal Get proceedings delegated from the Publisher agent, and the resulting dependency between the two agents. The delegated goal is AND-decomposed into sub-goals, which are either operationalised through 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.

Notice that there may exist alternative operationalisations of the system agents. For example, in Fig. 16, the Proceedings Manager can print proceedings either via postscript (ps) or pdf. These alternatives differ in terms of their contribution to soft-goals. While the postscript option is better in terms of contribution to soft-goal printer fidelity, the pdf option is better for what concerns portability. The decision about which option to select is delegated either to the detailed design phase or, envisioning an adaptive system, to run-time.

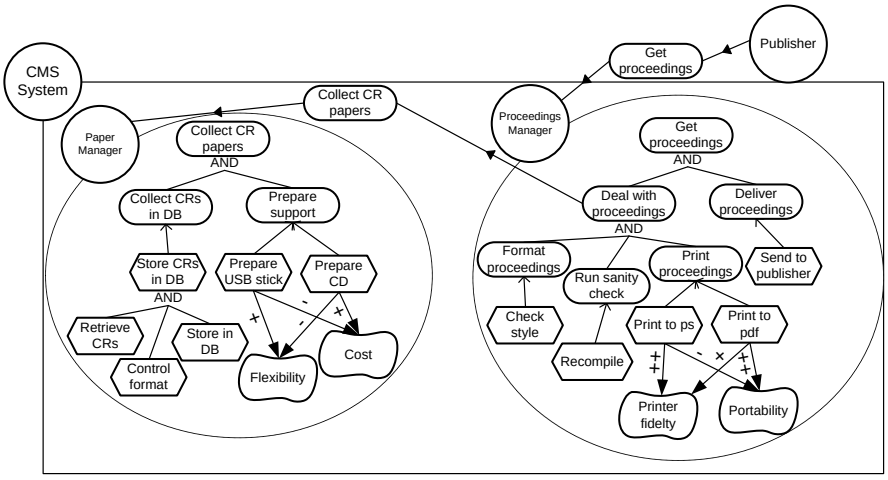


Fig. 16 Architectural Design: Simplified Goal Model of two agents of CMS.

6.2.4 Capability modelling

This activity details whether and how a specific agent is capable of achieving a goal by executing a specific plan. As suggested in [20], modelling capabilities includes assessing both ability and opportunity. Ability means that the agent can carry out a plan without interacting with (delegating to) other agents. A greater opportunity means that the plan contributes better than others to the stakeholders' preferences and QoS needs (i.e. to modelled soft-goals). 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. 16.

6.3 Work Products

Table 4 lists the work product types for the Architectural Design phase. The relationships of these work products, representing a *Tropos* goal model, with the metamodel elements, are described in Figure 8.

6.3.1 Work Products Examples

The AD Actor Diagram is illustrated in Fig. 16, which shows the delegations between the system actor and other agents. Fig. 15 compactly shows two AD Goal Diagram for agents "Paper Manager" and "Proceedings Manager". Table 5 illustrates an AD Capability Table.

Table 4 Work products for the Architectural Design phase.

Name	Description	Work Product Kind
AD Actor Diagram	It shows the dependencies from the system actor to the agents as well as the dependencies between the agents.	Structural
AD Goal Diagrams	They represent the goal diagram for each of the identified and analysed agents.	Structural
Capability Table	It lists all agent capabilities along with their contributions to soft-goals	Structured

Table 5 Example of an AD Capability Table.

Goal	Ability	Contribution
Prepare support	Prepare USB stick	Flexibility (+); Cost (-)
Prepare support	Prepare CD	Flexibility (-); Cost (+)

7 Detailed Design

The Detailed Design (DD) phase is concerned with the specification of the capabilities of the software agents in the system and of the interactions taking place, focusing on dynamic and input-output aspects, leading to a detailed definition of how each agent needs to behave in order to execute a plan or satisfy a goal. This includes the detailed specification of the single functionalities composing the plans associated to each agent's goals, the definition of interaction protocols and of the dynamics of the interactions occurring between agents and with systems and humans in the environment.

In this phase, also a decision on the implementation paradigm has to be made, either for going to a traditional object-oriented, or to an agent-based, goal-driven implementation. The activities in the phase are outlined in Figure 17 while Figure 18 shows the internal structure of the phase.

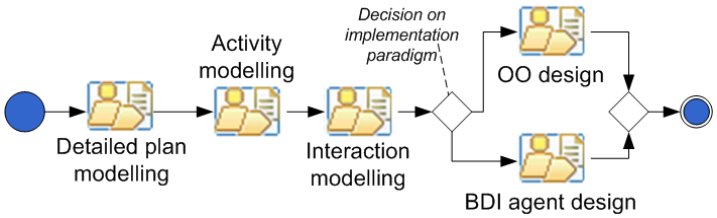


Fig. 17 Flow of activities in the *Tropos* Detailed Design phase. Iterations are not shown in the diagram, but possible starting at each activity.

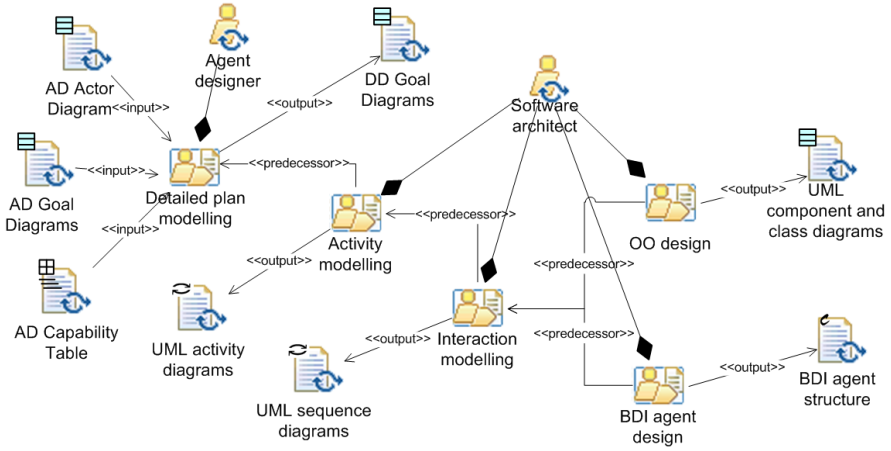


Fig. 18 The structure of the Detailed Design phase described in terms of activities, involved roles and work products. Please note that the internal use of work products in output of an activity as inputs of the subsequent activities is not shown explicitly, to keep the diagram concise.

7.1 Process roles

Two roles are involved, the *Agent Designer* and the *Software Architect*.

7.1.1 Agent Designer

The Agent Designer, involved also in the AD phase, is responsible for detailing the goal models obtained from the previous step, decomposing the plans into AND/OR hierarchies, till arriving to the concrete functionalities that compose each plan.

7.1.2 Software Architect

It is responsible for the detailed design of the system as a whole and of the single functionalities that have to be implemented. He has to take a decision for the programming paradigm to follow and the implementation platform to use, and to detail the models according to its decisions.

7.2 Activity Details

7.2.1 Detailed capability modelling

The agent designer details the capabilities identified in the AD phase through plan modelling by AND/OR decomposition to sub-plans, until reaching a fine-grained level defining the single activities that need to be available in the system.

7.2.2 Activity modelling

At this step, the software architect needs to take a first decision on the alternative capabilities and sub-plans which should be further specified and finally implemented, depending on the desired adaptability of the system and the affordable implementation effort. This selection should be made considering positive and negative contributions to softgoals which represent preferences and quality-of-service demands, or by employing a requirements prioritisation technique. If the aim is to create an adaptive system, a higher number of alternatives should be detailed and eventually implemented. To model the execution workflow for the activities that a capability is composed of, UML activity diagrams are adopted. UML activity diagrams can be directly derived from *Tropos* plan decompositions by model transformation, and further elaborated by detailing the dynamic aspects of a capability (see an example in Figure 19), including sequential and parallel workflows and alternative choices.

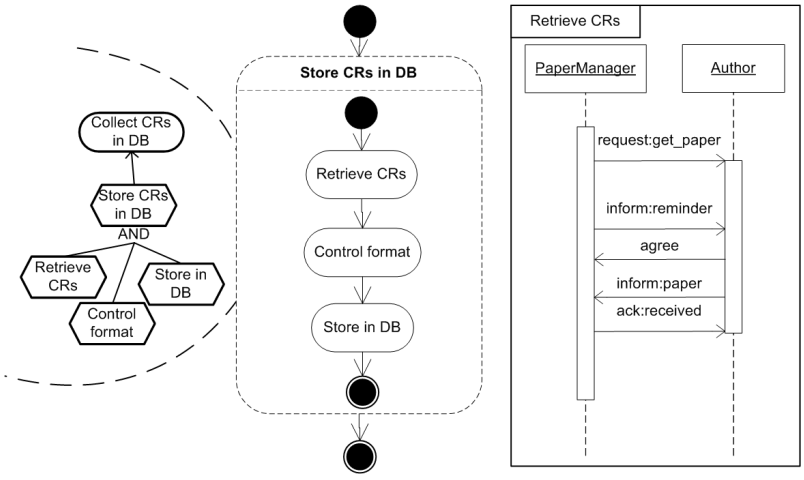


Fig. 19 Activity and interaction modelling: Goal diagram with a plan decomposition for a part of the CMS Paper Manager Agent, detailed in UML activity and sequence diagrams.

7.2.3 Interaction modelling

The software architect analyses the interactions that take place, between agents in the system which need to be detailed for each activity, and between the agents in the system and actors in its environment (including software agents and human actors). UML sequence diagrams (Figure 19) are used for specifying interaction protocols and details for each agent interaction and for single activities (i.e. leaf-level plans).

7.2.4 Platform-dependent design

At this stage, a decision on the implementation paradigm has to be made, either for going to a traditional object-oriented, or to an agent-based, goal-driven implementation. Following the central idea of *Tropos* to keep the notions of actor/agent and goal throughout all phases to avoid conceptual gaps, an agent-oriented implementation is recommended. Agents are autonomous entities with independent threads of control, that can interact with each other to reach their goals. Various frameworks are available, supporting these implementation concepts. An object-oriented implementation could be of benefit for performance-critical systems and for a better integration with existing software. In the following. we briefly touch on an object-oriented design, while the specific implementation phase proposed for *Tropos* relies on an agent-oriented design.

Object-Oriented design

At this point, a traditional OO approach can be followed for the detailed design, by using component and class diagrams for the description of the agents in the system and their relationships.

BDI agents design

Aiming at an agent-oriented implementation, after having modelled the single activities and interactions to be carried out by an agent (i.e. the *capability level*), in this activity the behavioural aspects, called the *knowledge level* [20], are detailed. This is achieved by mapping the high-level goal model of an agent, including dependencies and contribution relationships, into a generic BDI (belief-desire-intention) agent structure, according to a mapping such as the one defined in [19]. This structure can be enriched defining goal types (maintenance, achievement), goal satisfaction and goal failure criteria [15]. Selecting the proposed agent development platform Jadex, the mapping to an Agent Definition File specification is tool-supported [14].

7.3 Work Products

The Detailed Design phase generates four work products, which in part depend on the chosen implementation architecture, out of the ones listed in Table 6.

Table 6 Work products created during the Detailed Design phase.

Name	Description	Work Product Kind
DD Goal Diagram	It details the AD goal diagram of each agent, with the plans decomposed to sub-plans which represent the single functionalities that should be available.	Structural
UML Activity Diagram	It details a plan to the single activities it is composed of, and captures their temporal order.	Behavioural
UML Sequence Diagram	It explicitly shows the interactions between two software or human entities, possibly basing on interaction protocols, and defines the effects of an interaction.	Behavioural
UML Component and Class Diagrams	They illustrate the whole system by the use of UML diagrams, suitable if the target language is object-oriented.	Structural
BDI Agent Structure	It has a structure basing on the concepts of goal, plan and belief [3]. Adopting Jadex [22] as implementation platform, this structure is represented in an Agent Definition File.	Composite

7.3.1 Work Products Examples

Figure 19 illustrates the main work products of the Detailed Design phase. Left: an excerpt of a DD goal diagram with plan decomposition; middle: UML activity diagram for plan Store CRs in DB; right: UML sequence diagram detailing the interactions that need to take place in one of the leaf-level plans. Portions of a BDI agent structure are shown, implemented as Jadex Agent Definition File, on the right side of Figure 22.

8 Implementation and Testing

The last phase in the *Tropos* development process includes coding and testing activities, leading to the deployment of the final software.

Tropos does not impose the use of a specific implementation platform. However, it recommends the adoption of an agent-oriented language with the notions of agent, goal, and plan. In this way, conceptual gaps are reduced and the traceability of artefacts and decisions through the phases is simplified. Specifically, we describe an implementation on the *Jadex* [22] agent platform. Nevertheless, the activities described are general and apply in principle also to a development with other agent

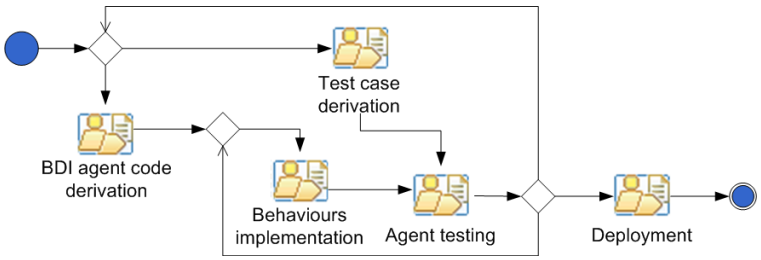


Fig. 20 Flow of activities for the Implementation and Testing phase.

languages, such as Jack, Jason or 2APL. Also, non-BDI agent languages such as JADE can be used for the full implementation, mapping goals to the artefacts available in the language.

The activities in this phase, outlined in Figure 20, are typically executed in an iterative way and, to some degree, in parallel, to take account of the revisions needed to reach to a final software product. Figure 21 shows the in- and output work products and the involved actors.

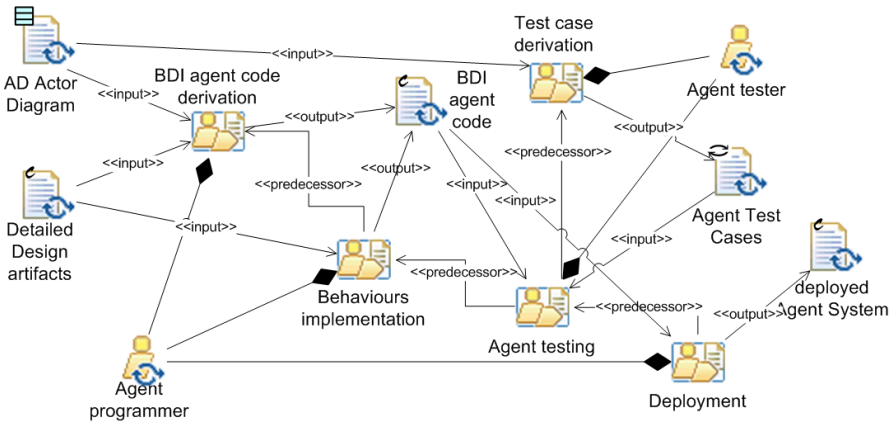
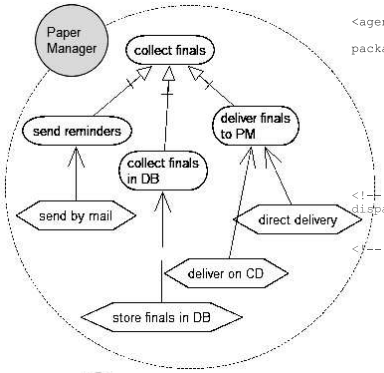
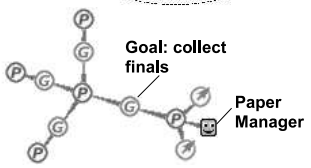


Fig. 21 SPEM model showing the structure of the Implementation and Testing phase with in- and output work products. The phase takes in input the AD actor diagram and the outputs of the DD phase.

8.1 Process roles

Two roles are involved: the *Agent Programmer* and the *Agent Tester*.






```
<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. 22 Left: Simplified goal diagram for PaperManager modelled using the *TAOM4e* tool. Right: part of the *Jadex* XML code generated with the *t2x* tool. Bottom: example *Jadex* run-time agent instance with activated goals and plans, visualized with the *Introspector* tool provided by the *Jadex* platform.

8.1.1 Agent Programmer

The Agent Programmer is responsible for carrying out the implementation of the agent system, starting from goal models and UML diagrams. Typically, he also carries out the deployment of the software system.

8.1.2 Agent Tester

On the basis of goal models and sequence diagrams, the agent tester derives test cases, which are then executed on the software agents.

8.2 Activity Details

We detail each of the activities of the phase and explain the artefacts involved.

8.2.1 BDI agent code derivation

The goal and UML models created in the DD phase and the AD actor diagram are the basis for the implementation of software agents. Selecting Jadex as a target platform and using the *t2x* tool, *Jadex* code can be generated basing on the BDI agent structures previously defined. 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. The generated code implements the agent's reasoning mechanisms needed to select correct plans at runtime to achieve desired goals defined in the agent's AD goal model. It has to be customized adding the temporal and operational aspects defined in the detailed design UML models, and implementing the interaction protocols. FIPA standard agent interaction protocols such as *Request* and *Contract Net* are predefined. As an example, Fig. 22 shows part of the generated *Jadex* code in XML format of the agent Paper Manager from the CMS example. This fragment of code corresponds to the *Tropos* goal model on the top-left side of the figure, and it reasoning trace at run-time is presented on the bottom-left corner of the figure.

8.2.2 Behaviours implementation

With the DD activity and sequence diagrams in input, in this phase the *behaviours* of the agents are implemented. The behaviours realise the plans defined in a goal model to operationalise the goals. Behaviours can be implemented with OO concepts (e.g. in JAVA) or using specific concepts present in languages such as *JADE* [2].

8.2.3 Test case derivation

A systematic way of deriving test cases from goal-oriented specifications and techniques to automate test case generation and their execution has been introduced in [18]. Such approach considers different testing levels, from unit testing to acceptance testing, and different aspects of testing a Multi-agent system that adopts *Tropos* design. Test cases are derived from the agent specifications (specifically, the AD actor diagram) with the aim to test and validate the interactions between agents (represented as dependencies in *Tropos*) and the achievement of goals, adopting domain-specific metrics. Test cases derivation is supported by a semi-automatic tool [17], providing a GUI-based editor to detail test scenarios and inputs.

In the case of the CMS, the *eCAT* testing tool [17] takes the architectural diagram in Figure 15 as an input and generates a set of test suites for each agent.

8.2.4 Agent testing

The agent testing activity consists of an automated testing of the agents in a virtual environment, observing the interactions with the environment and with the peer agents, while varying environment and inputs for each test case. This activity can be automated and parallelised with the help of the testing framework introduced in [18].

8.3 Deployment

The BDI agent code, including the implemented behaviours, can be executed on the *Jadex* platform. Interactions and the goal achievement process can be visualized via tools provided by the platform. Regarding the present case study, code was generated for the two system agents ProceedingsManager and PaperManager. As an example, Fig. 22 shows an excerpt *Jadex* code in XML format, for the agent Paper Manager. This fragment of code corresponds to the *Tropos* goal model on the top-left side of the figure, and it reasoning trace at run-time is presented on the bottom-left corner of the figure.

8.4 Work Products

Focusing on an agent-oriented implementation, the implementation and testing phase generates four main work products, listed in Table 7.

Table 7 Work products created during the Implementation and Testing phase.

Name	Description	Work Product Kind
BDI agent code	It represents the executable agent code that exhibits the behaviour defined in the previous phases.	Composite
Agent test cases	They define a list of inputs and corresponding outputs (data, interactions, exhibited behaviours), to test the correctness of the single agent implementations and of the whole system. Defined in XML format, they can be used as input for automated testing agents.	Behavioural
Deployed agent system	It represents the running agent system.	Composite

8.4.1 Work Products Examples

The BDI agent code depends on the selected implementation platform. In Jadex, it corresponds to an Agent Definition File, such as on the right side of Figure 22, for

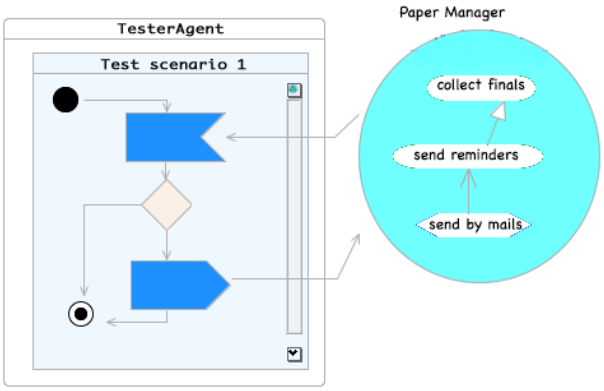


Fig. 23 An example of a test case for the Paper Manager, specified using the eCAT editor.

each agent, in combination with Java code. The behaviour of the deployed agents can be visualized at run-time on the agent platform, such as in the lower left part of Figure 22. Figure 23 depicts and example of a test case that checks the goal send reminders of the Paper Manager. The test case is specified using the eCAT editor; following the test scenario of the test case, the Agent Tester waits for a message from the agent under test (Paper Manager), checks for the content of the message, and sends back a notification.

9 Work Product Dependencies

The work product dependency diagram in Figure 24 describes the dependencies among the different work products created in the five development phases. Focusing on an agent-oriented implementation, an eventual object-oriented implementation is not considered.

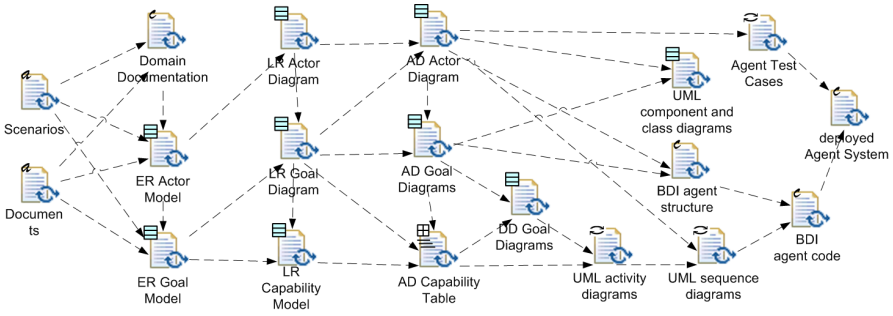


Fig. 24 Work product dependency diagram for the *Tropos* methodology.

References

1. Yudistira Asnar, Paolo Giorgini, and John Mylopoulos. Goal-driven risk assessment in requirements engineering. *Requir. Eng.*, 16(2):101–116, 2011.
2. Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE: A FIPA Compliant agent framework. In *Practical Applications of Intelligent Agents and Multi-Agents*, pages 97–108, April, 1999.
3. Lars Braubach, Alexander Pokahr, Daniel Moldt, and Winfried Lamersdorf. Goal representation for bdi agent systems. In *PROMAS*, pages 44–65, 2004.
4. 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.
5. Volha Bryl, Fabiano Dalpiaz, Roberta Ferrario, Andrea Mattioli, and Adolfo Villafiorita. Evaluating procedural alternatives: a case study in e-voting. *EG*, 6(2):213–231, 2009.
6. Volha Bryl, Paolo Giorgini, and John Mylopoulos. Designing cooperative IS: Exploring and evaluating alternatives. In *OTM Conferences (1)*, pages 533–550, 2006.
7. L.K. Chung, B. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Publishing, 2000.
8. 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.
9. 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.
10. Paolo Giorgini, Fabio Massacci, John Mylopoulos, and Nicola Zannone. Modeling security requirements through ownership, permission and delegation. In *Proceedings of the 13th IEEE International Requirements Engineering Conference (RE'05)*, 2005.
11. Irit Hadar, Tsvi Kuflik, Anna Perini, Iris Reinhartz-Berger, Filippo Ricca, and Angelo Susi. An empirical study of requirements model understanding: *Use Case* vs. *tropos* models. In *SAC*, pages 2324–2329, 2010.
12. M. Kolp, P. Giorgini, and J. Mylopoulos. A goal-based organizational perspective on multi-agents architectures. In *Proceedings of the Eighth International Workshop on Agent Theories, architectures, and languages (ATAL-2001)*, 2001.
13. Mirko Morandini, Duy Cu Nguyen, Anna Perini, Alberto Siena, and Angelo Susi. Tool-supported development with tropos: The conference management system case study. In Michael Luck and Lin Padgham, editors, *Agent Oriented Software Engineering VIII*, volume 4951 of *LNCS*, pages 182–196. Springer, 2008. 8th International Workshop, AOSE 2007, Honolulu, HI, USA, May 2007.
14. Mirko Morandini, Loris Penserini, and Anna Perini. Automated mapping from goal models to self-adaptive systems. In *23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), Tool Demo*, pages 485–486, September 2008.
15. Mirko Morandini, Loris Penserini, and Anna Perini. Operational Semantics of Goal Models in Adaptive Agents. In *8th Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'09)*. IFAAMAS, May 2009.
16. Mirko Morandini, Anna Perini, and Alessandro Marchetto. Empirical evaluation of tropos4as modelling. In *iStar*, pages 14–19, 2011.
17. Cu Duy Nguyen, Anna Perini, and Paolo Tonella. ecat: a tool for automating test cases generation and execution in testing multi-agent systems. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: demo papers*, pages 1669–1670. International Foundation for Autonomous Agents and Multiagent Systems, 2008.
18. Cu Duy Nguyen, Anna Perini, and Paolo Tonella. Goal-oriented testing for MASs. *International Journal of Agent-Oriented Software Engineering*, 4(1):79–109, 2010.
19. Loris Penserini, Anna Perini, Angelo Susi, Mirko Morandini, and John Mylopoulos. A Design Framework for Generating BDI-Agents from Goal Models. In *6th Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'07), Honolulu, Hawaii*, pages 610–612, 2007.

20. Loris Penserini, Anna Perini, Angelo Susi, and John Mylopoulos. High variability design for software agents: Extending tropos. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 2(4), 2007.
21. 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.
22. Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. Jadex: A bdi reasoning engine. In J. Dix R. Bordini, M. Dastani and A. El Fallah Seghrouchni, editors, *Multi-Agent Programming*, pages 149–174. Springer Science+Business Media Inc., USA, 9 2005. Book chapter.
23. Angelo Susi, Anna Perini, John Mylopoulos, and Paolo Giorgini. The tropos metamodel and its use. *Informatica (Slovenia)*, 29(4):401–408, 2005.
24. E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, Department of Computer Science, University of Toronto, 1995.