A Goal-Oriented Approach for Modelling Self-Organising MAS

Mirko Morandini[1], Frédéric Migeon[3], Marie-Pierre Gleizes[3], Christine Maurel[3], Loris Penserini[2], and Anna Perini[1]

[1]FBK IRST, Via Sommarive 18, I-38100 Trento, Italy
[2]University of Utrecht, Padualaan 14, Utrecht, The Netherlands
[3]IRIT, 118 Route de Narbonne, 31062 Toulouse, France
{morandini|perini}@fbk.eu, loris@cs.uu.nl,
{migeon|gleizes|maurel}@irit.fr

Abstract. Autonomous software agents provide a promising solution to the needs of decentralised networked systems, able to adapt their behaviour in a complex and dynamically changing environment.
Current agent-oriented software engineering methodologies tend to focus on different levels to realise such a self-adapting behaviour, namely the agent individual level and the global system level. The first requires to design a goal-directed agent behaviour, the second to design agents able to optimize their coordination with other peer agents in the organization, giving rise to system-level adaptation.
In this paper we propose to extend a goal-oriented engineering methodology to deal with the modelling of organisations that are able to self-organise in order to reach their goals in a changing environment. To deliver on this aim, we combine *Tropos4AS*, an extension of *TROPOS* for adaptive systems, with concepts, guidelines and modelling steps from the *ADELFE* methodology, which provides a bottom-up approach for engineering collaborative multi-agent societies with an emergent behaviour.
The resulting MAS has self-adaptation properties, having agents that are able to change their behaviour according to changes in the environment, and having organisations that adapt themselves to changing needs. The approach is illustrated by modelling a collaborative multi-agent system for conference management.

1 Introduction

Nowadays, networked systems require decentralized and flexible configurations, which are able to support mediated services (e.g. flight booking systems) as well as peer-to-peer business/social relationships (e.g. eBay, social networking). Such software systems need to exhibit an increasing level of self-adaptivity, at the system or component level, in order to operate efficiently in a dynamically changing environment. Engineering such decentralized, self-adapting networked systems poses challenging issues for the software engineering methodologies research.

Autonomous software agents have been largely studied for their property to exhibit self-adaptive behaviours at different levels. At the level of the individual agent, the agent has ability to perceive the surrounding environment, to interpret collected information

and to reason on it. This enables it to decide which behaviour to adopt in a context-aware manner. At the level of global (multi-agent) system, agent cooperation mechanisms can give raise dynamically to an emergent behaviour of the system.

Agent-Oriented Software Engineering methodologies propose methods and techniques to build this type of systems, but typically they tend to focus on a specific level of self-adaptivity. So, for instance the *TROPOS* methodology [3], and its extension for self-adaptivity *Tropos4AS* [13], focus on building software agents that behave in a goal-directed way and are able to dynamically switch from a behaviour to another one to avoid failure in achieving their own goals or to better meet quality requirements. On the other side, *ADELFE* [2] is an agent-oriented methodology tailored to engineer self-adaptive multi-agent societies by cooperative agents based on the AMAS theory [1]. *ADELFE* enables with a bottom-up approach to define the nominal and cooperative behaviour of the agents which leads to self-organisation and system's self-adaptation.

We study here how to engineer agent systems which can adapt autonomously to a changing environment. This system should exhibit a self-adaptive behaviour of single agents as well as a self-organising behaviour of the agent society.

In this paper, we investigate benefits from extending the *Tropos4AS* agent modelling framework with *ADELFE* modelling activities, along two main lines: *1)* extending the *Tropos4AS* modelling language meta-model by including concepts from the *ADELFE* meta-model; and *2)* revisiting the *Tropos4AS* design process by including *ADELFE* activities. For illustration, the process is applied to a conference management system example, giving a first evidence for its benefits.

The paper is organised as follows: in Section 2, the two methodologies *TROPOS* with the *Tropos4AS* extension, and *ADELFE* are briefly recalled and compared. Section 3 describes the *Tropos4AS* extension at the level of the meta-model and at the level of the design process. Section 4 illustrates the application of the resulting design approach to the conference management system example and discusses main findings from this experience. Related work is presented in Section 5, conclusions and future work in Section 6.

2 Methodological Background

2.1 Tropos and Tropos4AS: Goal-Oriented Modelling

The agent-oriented software engineering methodology *TROPOS* [3] adopts ideas from the MAS paradigm and from *i**, an organizational modelling framework for requirements analysis, founded on the 'mentalistic' notions of actor, goal, softgoal, task, resource, and social dependency between actors. The *TROPOS* modelling language is used along the whole development process: in Early Requirements Analysis (ER) the human organisational settings in which the system will be used are analysed; in Late Requirements Analysis (LR) the system-to-be is introduced and its requirements are modelled in terms of dependencies between stakeholders and the system itself; in Architectural Design (AD) the system actor is decomposed into components (*sub-system actors*), each having responsibility for the achievement of a part of the system's goals;

in Detailed Design (DD) the system components and their interactions are further specified. *The modelling process follows a top-down approach, refining the global goals, which were delegated from the stakeholders to the system-to-be.*

For the aim of this work, we give more details on the LR and AD phases. At the end of the LR phase, the system's requirements are modelled by delegation of goals and tasks from the stakeholders and users in the organisation, to the system, itself modelled as an actor. The delegated goals are then organised in the system's goal model, decomposed into more concrete goals and possibly operationalised by plans. Subsequently, as key step in AD, the system actor is decomposed into sub-actors, which will become the software agents at the implementation (in Fig. 1). This decomposition is usually done by distributing system functionalities to single subactors and gathering similar functionalities. Different architectural patterns can be applied to it, e.g. [4]. As next step, the single sub-agents' goal models are detailed, by goal analysis, decomposition and delegation.

Tropos4AS [13] extends *TROPOS* goal models to enable the description of systems that shall be able to adapt to environmental changes. *Tropos4AS* introduces modelling of an actor's perceived environment and of possible failures that can be identified and prevented by recovery activities. Moreover, goals can be annotated to define the runtime goal achievement behaviour, with various goal types and conditions related to the environment, for goal creation, achievement and failure. The resulting design model is mapped into implementation language constructs to derive a code skeleton.

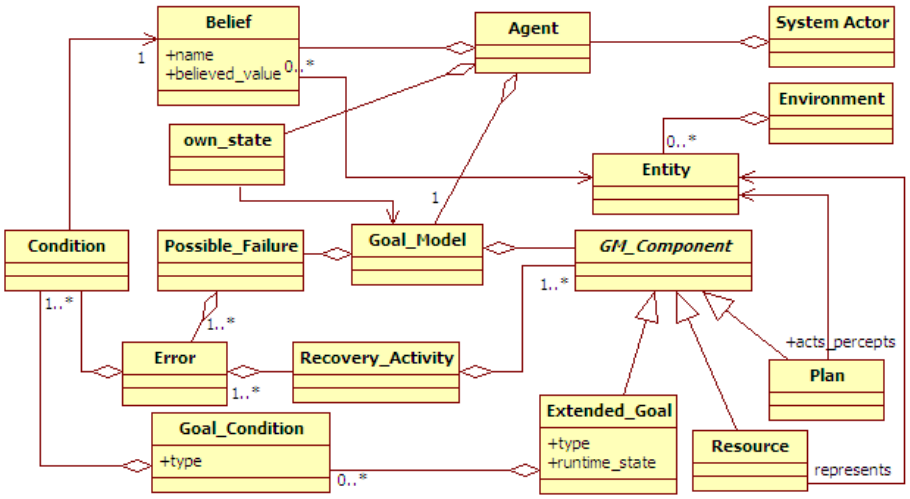

Fig. 1. The Tropos4AS metamodel, showing the composition of an agent (simplified view of the goal model).

Fig. 1 shows details from the *Tropos4AS* metamodel, illustrating the composition of a single *Tropos4AS* agent, an actor identified as sub-system actor at *TROPOS* detailed design.

2.2 ADELFE: Cooperative Agents

ADELFE[1] is an agent-oriented methodology for designing Adaptive Multi-Agent System (AMAS) [1]. MAS developed according to *ADELFE* provide an emergent global function [10]. It is qualified as emergent because it is not coded inside the agent. To obtain this emergent behaviour, the AMAS theory [5] propones to design agents with the ability to autonomously and locally modify their interactions in order to react to changes in their environment. This system is self-organising and is able to adapt to its environment. According to the AMAS principles, interactions between agents depend on their local view and on their ability to "cooperate" with each other. Every internal part of the system (agent) pursues an individual objective and interacts with agents it knows by respecting cooperative techniques which lead to avoid or to remove Non Cooperative Situations (NCS) like conflict, concurrence etc. Facing a NCS, a cooperative agent acts to come back to a cooperative state and permanently adapts itself to unpredictable situations while learning on others.

The *ADELFE* methodology covers the phases of usual software design from the requirements to the implementation, with the addition of specific activities to support the design of adaptive multi-agent systems.

The modelling process follows a bottom-up approach, defining the cooperation rules and activities of the single agents, leading to an emergent behaviour of the system.

In the analysis phase, *ADELFE* gives guidelines to decide if the application is adapted to an implementation following the AMAS principles. Furthermore, it provides guidance to identify cooperative agents among all the entities defined during the final requirements. Concerning the design phase, three activities are added. The first concerns the relationships between agents. The second is about the agent design. In this activity, the cooperation failures are defined. Third, a fast prototyping activity helps to build and verify the agent behaviour. Moreover, the implementation phase uses Model-Driven Engineering principles to produce code skeletons for the MAY[2] middleware.

The design phase is based on a AMAS metamodel characterising as precisely as possible the concepts involved in the AMAS principles and mandatory for *ADELFE* such as Perceptions, Actions, Aptitudes, Cooperations Rules, Non Cooperative Situations, Representations, Skills, etc. Figure 2 shows that a cooperative agent is defined with a *PerceptionModule*, a *DecisionModule* and an *ActionModule*. Decision is implemented with *Rules* tha trigger an *action* or a *skill*. These rules describe either standard behavior (with *StandardRules*) or cooperation (with *CooperativeRule*) according to *NonCooperativeSituation* type.

2.3 Comparing the Two Methodologies

Up to now, in this section we described two AOSE methodologies founding on very different principles and having a different scope. While *ADELFE* is tailored to decentralised, adaptive complex systems and follows a bottom-up approach to eventually

[1] ADELFE is a French acronym for "Atelier de Dveloppement de Logiciels Fonctionnalit Emergente", see `http://www.irit.fr/ADELFE`

[2] `http://www.irit.fr/MAY`

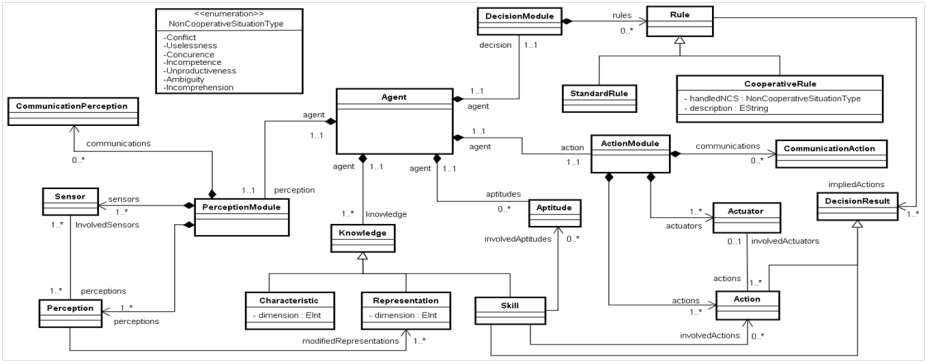

Fig. 2. Portion of the ADELFE metamodel concerning the cooperative agent.

reach the global goal of the system in an emergent way through agent cooperation, *TROPOS* claims to be a general methodology, where the system goals elicited by analysing the organisational settings, through steps of refinement and decomposition lead to program components implementable in software agents.

Albeit in both methodologies, agents are a metaphor for an autonomous entity with own goals and abilities, trying to achieve their local goals, the process of obtaining the single agent's goals presents conceptual differences: *ADELFE* agents are identified and their behaviour specified, analysing the domain entities, their role in the system and the relationships between them. They create a complex organisation by having at run-time a high number of instances for each agent type. The global goal of the system, which the stakeholders want to obtain from the software system, is modelled in use cases, but this global goal is not coded by the single agents and can only be observed, emerging from the collective behaviour.

The *TROPOS* development process starts with requirements elicitation and analysis, to capture the objectives of the stakeholders. The MAS architecture is then obtained by analysing the organisational settings with the goals and tasks delegated to the system, decomposing them and delegating their satisfaction to single roles or agents, following general engineering rules to achieve low coupling and high correlation between the tasks to be achieved by a single agent.

Next, the structure and abilities of the single agents have to be defined. In *ADELFE*, a central role is given to agent interaction and coordination, specifying behaviour rules and associated activities both for the agent's nominal behaviour (i.e. the ordinary behaviour exhibited by the agent in a working situation without problems and failures) and its cooperative behaviour (Especially focusing on how to react to collaboration problems). Moreover, the agent's belief (representation) of the outside world, and its sensors and actuators are defined.

TROPOS agents are characterised both by the goals delegated from the stakeholders, and the dependencies to other agents; the nominal behaviour is defined by its goal model, which includes plans to perform and resources to provide, to achieve goals.

By the *Tropos4AS* extensions, the goal runtime behaviour can be further specified, defining goal types and conditions on to the environment perceived by the agent. Exceptional behaviour can be defined by modelling possible failures, causing errors, and recovery activities.

Finally, In the design phase, in both *TROPOS* and *ADELFE*, further details on the implementation can be given by UML2.0 diagrams.

3 Modelling of Self-Organising MAS

Tropos4AS follows a top-down approach from the system to the single agents and their behaviour, and achieves traceability by decomposition and delegation of goals through the design phases.

TROPOS requirements modelling is prominent for it's ability to capture the organisational settings where the system to develop will be integrated and the dependencies and responsibilities of the agents in the system and the actors playing different roles in the organisation.

However, *TROPOS*, as well as *Tropos4AS*, lack of support for agent organisations, i.e. for modelling the dynamics of collaboration between software agent instances in a multi-agent organisation where each modelled agent has various instances, which can also be dynamically added and removed.

The *ADELFE* methodology was created specifically for the development of such agent organisations. However, it adopts a bottom-up approach, to achieve the system's goal in an emergent way; the relationship between global goal and single agent's behaviour is not modelled and the global goal can only be observed from action and interaction of the parts.

Integrating ideas and modelling steps from *ADELFE* we enrich *Tropos4AS* for the modelling of agent organisations.

A bottom-up addition of *ADELFE* cooperation rules (which fit well into the actual concept of failure modelling) will give to the run-time agent instances the knowledge for selection of and cooperation with their peers, and thus achieve an emergent self-organising behaviour to adapt to a changing environment.

3.1 Metamodel Extension

Here, we investigate how to extend the *Tropos4AS* meta-model with concepts taken from the *ADELFE* meta-model, and revise the *Tropos4AS* design process including steps that belong to the *ADELFE* approach.

To improve modelling the interplay of an agent with the entities and actors inside and outside the software system under development, we explicitly add the concept of agent's knowledge about itself and about its environment.

ADELFE provides modelling of the agent's knowledge by characteristics (facts the agent is sure about), representations of the environment as perceived through sensors, and the agent's skills (Fig. 2). We integrate characteristics and representations (corresponding to the agent's belief) in the extended model. Information captured by *Skills*, *Aptitudes*, the agents *Actions* and its nominal behaviour, encoded in *Rules*, is mainly

covered by the *TROPOS* goal model, a main component of the *Tropos4AS* metamodel (Fig. 1).

Understanding the interplay between the agent and its environment is of major importance to understand and model a system's self-adaptivity. To model the agent's environment we integrate ideas from the *Agents and Artifacts* approach [15]. Artifacts represent the non-intentional entities (*ADELFE* passive entities), in- and outside the boundary of the system to develop. They provide an interface to the external world, to the users and also to other agents, through *social artifacts* such as a whiteboard or a communication channel.

The extended metamodel is shown in Fig. 3 and defines an agent in the system (represented as a *TROPOS* system actor) with its components: goal model, knowledge (the "belief base"), the system and the external environment.

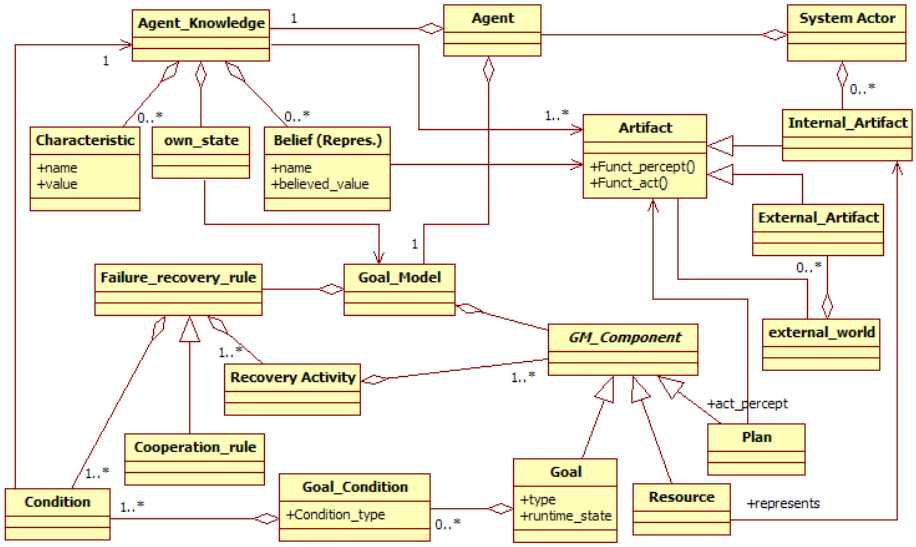

Fig. 3. Metamodel for adaptive, cooperative agents, which extends the TROPOS4AS meta-model with ADELFE concepts (simplified view of the goal model).

Regarding the aim of this paper, the most important step from *ADELFE*, integrated in the methodology, will be the elicitation of non-cooperative situations and modelling of the discovered so-called cooperation rules. We enhance *Tropos4AS* failure modelling by allowing to specify failure recovery with rules (class Failure_recovery_rule in Fig. 3). Cooperation rules are considered s specialization of failure recovery rules, with a well defined scope. Such failure recovery rules are composed by conditions to the agent's knowledge (on itself and its environment) and by recovery activities composed by represent a goal model fragment — a single *TROPOS Plan* (which corresponds to an *ADELFE Action*) or a more complex goal model.

3.2 Modelling Steps

We adapt the *TROPOS* modelling process to modelling of the newly introduced concepts. The proposed modelling steps are placed after the *TROPOS* Early (LR) and Late Requirements (LR) phases, described in [17]. As result of the LR phase, the requirements are modelled in terms of strategic dependencies between stakeholders, such as users and other external actors, to the software system, which is also modelled as an actor. The system actor has its own goals, plans, resources which are derived along these dependencies. This model is given as input to the following modelling steps.

Step 1 With the LR model in input (an example in Fig. 4), define the system from the *ADELFE* viewpoint (*ADELFE* activity 12): identify passive and active entities in- and outside the system to develop, and identify from the active entities the autonomous agents participating in the collective task. Output: an AMAS-ML System-Environment diagram (Fig. 5).

Step 2 In the *TROPOS* AD phase, guide the decomposition of the system actor identified in the LR diagram, into sub-actors (the agents in the system), according to the agents identified in the AMAS-ML system-environment diagram. The *TROPOS* system will include agents participating to this global task and agents achieving non-collective goals delegated by some stakeholder, or that have to supervise the collective task. The actors participating in self-organisation are highlighted (Fig. 6).

Step 3 With the *TROPOS* model resulting from *Step 2* in input, detail the high-level **nominal behaviour** of the single agents in the system by defining their goal and plan dependencies, and detailing their goal models by *TROPOS* goal analysis, until finding the plans to achieve the goals. The environment perceived by the agent is modelled considering the passive entities identified in the previous step, and the resources modelled in Tropos LR. From the dependencies and interactions between entities, the perception and action functionalities of the artifacts in the environment can be identified. Beliefs describe the agent's perception of these artifacts. This step is no more detailed here as it is not central to self-organisation.

Step 4 With the *Tropos4AS* model of Step 3 in input, which includes the dependencies between agents, focus on the collective task and define the necessary interactions, following *ADELFE* activity 13. Give special attention to failures that can arise from perturbations in the interaction between agents (which are cooperative by definition). The **exceptional behaviour** of each agent is now detailed by identifying non-cooperative situations that can arise. It is captured by conditions on the agent's knowledge together with the recovery activities to execute (an example in Table 1). These rules guide the single agent's self-organising behaviour, with activities that can be categorised in three groups: change of the own behaviour (*tuning*), change of partnership (*reorganisation*), and creation/deletion of agents (*evolution*).

Next, following the *Tropos4AS* process, the goal model built in step 3 can be detailed, adding conditions, goal types and relationships, to define a more detailed nominal behaviour, and modelling possible failures not ascribed to collaboration. Modelling can continue with *TROPOS* Detailed Design (DD), detailing plans (*capability level*)

and low-level interactions by UML diagrams [17], obtaining models that can be directly used as input for the implementation phase. For example, following a mapping as in [12], goal models can be mapped to *Jadex* agent code, artifacts to Java classes and failure conditions, including cooperation rules, to goal conditions.

4 Application to an Example

The design process is shown on a conference management system (CMS) example, described in [6], a case study used several times for agent systems developed with different agent-oriented software engineering methodologies [7].

A conference management system involves several stakeholders and has to satisfy users playing various roles, such as authors, reviewers, program committee members and the publisher. In the submission phase, authors need to be supported, and subsequently, *R4P* suitable reviewers have to be found for each paper, distributing the workload evenly. For this, each paper is described by *KP* keywords providing its main expertise area. Each reviewer describes its expertise fields with *KR* keywords and should review at most *P4R* papers.

Reviews have to be collected and evaluated to decide about acceptance or rejection of each submission, and finally the authors have to be notified, and the corrected camera ready papers collected and formatted. The prepared proceedings have then to be handed out to the publisher for printing. Fig. 4 shows the corresponding *TROPOS* LR diagram.

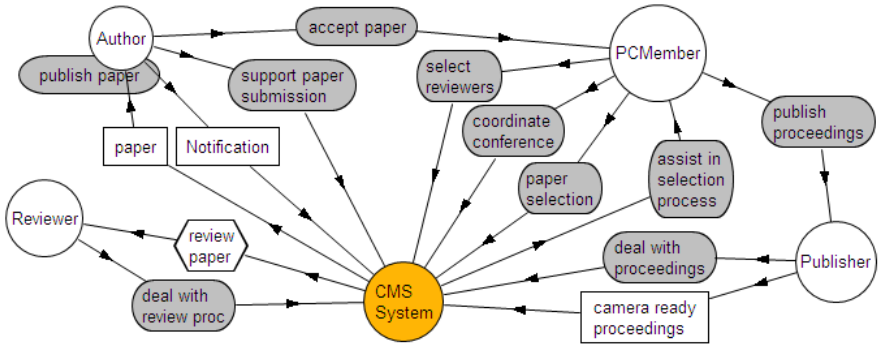

Fig. 4. *TROPOS* Late Requirements (LR) analysis: Definition of the system's objectives. Notice, that dependencies between actors entail a flow of information in the opposite direction.

We want to obtain a system composed by agents associated to each physical entity or role that has the need of autonomous decision and interaction, e.g. one for each paper, reviewer, etc. These agents are not "personal agents" acting selfish for the benefit of their relative stakeholder, but agents belonging to the system that are trusted and cooperative.

Interesting phases from the point of view of self-organisation between agents (which will then result in a system-mediated collaboration between physical actors or entities)

are the assignment of papers to reviewers, the collection of reviews and the decision of paper acceptance. We focus on the scenarios involving the reviewers. The reviewing process can possibly also be exposed to different kinds of unwanted perturbations. For example, unavailable reviewers, an unbalanced amount of papers in a particular area with a small number of competent reviewers, or withdrawn for any reason. Despite these eventualities could, in this small example, also be handled deterministically, they give a good example to show how a robust system should self-adapting, to meet its objectives. We now show the modelling process, going through the steps defined in Section 3.2.

4.1 Architecture

Following Step 1, we analyse the diagram in output of the *TROPOS* LR phase (Fig. 4). We identify 7 active and 2 passive entities. The active entities participating in the system's collective task are the `paper` and `reviewer agents`, representing the single submitted papers and the reviewers (Fig. 5).

We give to the PC chair agent – an agent in *TROPOS*, and an active entity in *ADELFE* (but not one participating to the collective task) – the charge to observe the society and to decide when a stable and optimal state is reached, in which all papers are assigned to reviewers. It will also be able to advise reviewer agents to relax some constraints (e.g., allocation of more than P4R papers per reviewer).

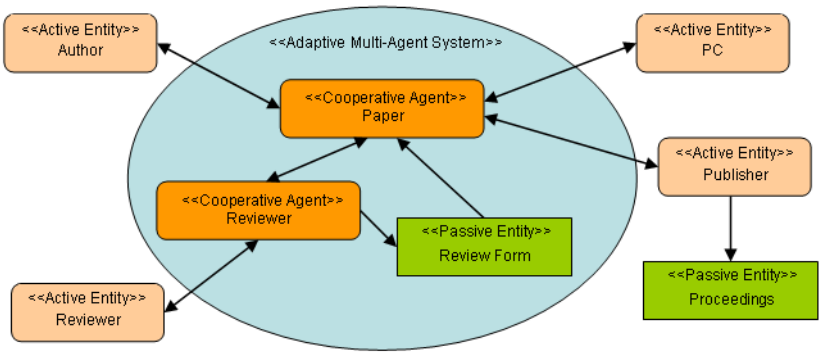

Fig. 5. Adelfe system-environment diagram showing the participating entities and the cooperative agents, inside the system boundary, related to the review assignment scenario.

Guided by the agents and active entities identified in Fig. 5, following Step 2 we decompose the `CMS system` in (Fig.4) into four sub-actors: `paper agent` and `reviewer agent`, which take part in the collective task of paper-review assignment, will be associated to the single physical papers and reviewers. The `program chair`

agent and the `proceedings agent` get their goals delegated from the physical actors playing the respective role in the organisation where the system is deployed and have thus also to be part of the software system (Fig. 6).

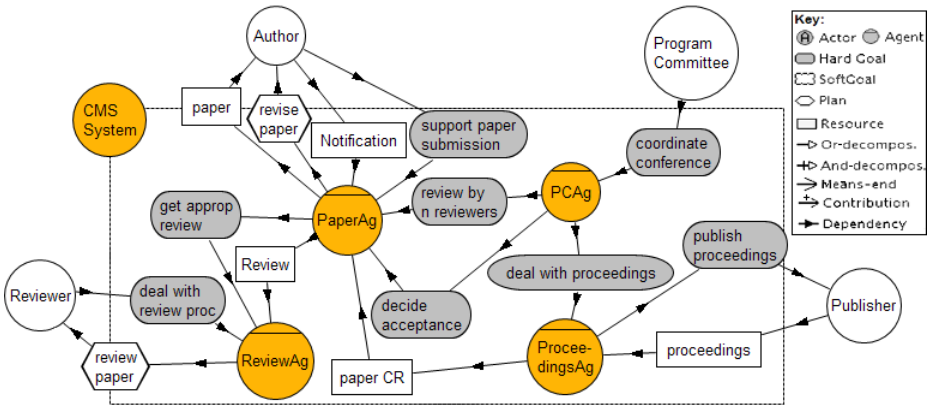

Fig. 6. TROPOS diagram of the multi-agent architecture.

4.2 Detailed design

In Step 3, the goals delegated from the stakeholders to the system are refined in the goal models of each sub-actor. Goals are decomposed until they can be operationalised by plans. Also, new dependencies between the different sub-actors arise (Fig. 7).

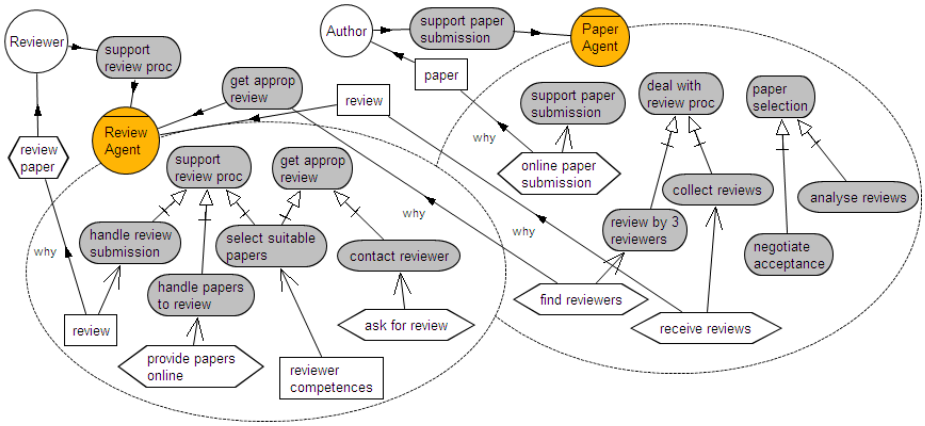

Fig. 7. Details form the goal models of the sub-actors paper and reviewer. Key in Fig. 6.

Tropos4AS provides the means for capturing the nominal goal achievement behaviour, defining when a goal will be activated, achieved, or dropped, capturing its representation of the environment and linking its execution to environmental changes. For example, the goal `get approp review` is created after *R4P* reviewers were assigned to a paper; achieved when *R4P* reviews are collected; and failed if a review is missing at the deadline.

Individual goals/objectives of agents In order to give a detailed view, we limit to the scenario of paper assignment to reviewers. Without a centralised distribution of papers to reviewers, the relative agents have to find a relevant allocation between papers and reviewers by self-organising to achieve an optimal distribution of papers and a timely collection of appropriate reviews, being robust for possible perturbations.

In Step 4, from the *TROPOS* model in output of step 3, we refer the local goals and the interactions (goal, task and resource dependencies) of the agents participating in the collective task, whose details will now be further modelled following the *ADELFE* process.

In order for any paper to meet reviewers, we design the system environment as a big room (a grid in practice) where reviewers can stand on at most one square. Paper agents can move on it to find matching reviewers. This approach was already experimented for a dynamic time-tabling elaboration with good results [18]. Furthermore, we define the notion of criticality of the `paper agent`, which is a criteria to know which one is under greatest number of constraints. It describes its difficulty to find a reviewer; it corresponds to the number of reviewers who have been met but are not relevant.

Nominal behaviour: Reviewer agents are placed on the grid and don't move. Paper agents are initially placed randomly on the grid and move in order to find reviewers and store their place under some constraints such as a reviewer cannot accept a paper if he is also author of the paper. Each paper agent remembers last *N* reviewer agents that it met, where it met them and what are the keywords associated to each.

Cooperative behaviour: The behaviour for the cooperation between the agent instances at run-time is defined by defining the agent's reaction to situations that are recognised to be "non-cooperative". This behaviour is expressed by collaboration rules containing activating conditions and associated recovery activities. Table 1 contains possible non-cooperative situations, and the collaboration rules, composed by a state and conditions, and the recovery activity to perform.

Take the example of the paper agent (Non-cooperative situation *PaperNCS3* in Table 1): If a paper finds a reviewer that fits to its keywords but is already associated to P4R papers, the less critical of them is asked to find a new reviewer (reviewer conflict). So, if a paper agent is very critical and adequacy (keywords matching) is not null, the association with the reviewer must be established. At the reviewers side, when a paper agent arrives, adequacy is computed. If matching is not obtained, the reviewer gives hints for other reviewers in its neighbourhood which could have enough matching keywords.

To conclude self-organisation at a point that a suitable configuration is achieved, the (single instance) `PC chair` agent observes the papers, which expose their criticality and their state, ranging from satisfied to unsatisfied.

Name	State	Description	Conditions	Recov. Activities
PaperNCS1	Exploration	A paper is getting closer to a reviewer already busy with another paper	Searching reviewer	Move in a different direction to find another reviewer
PaperNCS2	Exploration	Two reviewers are perceived	One of them is already busy	Move towards the reviewer that is free
PaperNCS3	Reviewer conflict	A paper found a reviewer that is already associated to *P4R* papers	Reviewer associated to *P4R* papers	Ask the less critical paper to search for another reviewer
PaperNCS4	Highly Critical	Paper agent is very critical and adequacy with reviewer is not null (but $< KP$)	High criticality and $0 <$ adequacy $< KP$	Association with reviewer is concluded
RevNCS1	No Matching	Matching keywords with an arriving paper is not obtained	Not enough matching keywords	Reviewer gives links to relevant neighbour agents
RevNCS2	Search Promotion	Reviewer agent promotes "mutual search" by asking paper agent what reviewers were already met	No matching keywords between the two	Remember paper agent's reviewers met.

Table 1. Description of main NCS for Paper-agents and Reviewer-agents. They will be modelled by cooperation rules

4.3 Discussion

The resulting design can be compared with a standard *TROPOS*, Prometheus and O-MaSE design of the CMS, as published in [7]. Despite it is divided into different agents, the *TROPOS* architecture achieved by a top-down decomposition of the system to subsystems is centralised and not a MAS of collaborating agents, as specified also in the requirements. For the same example, also the Prometheus methodology provides a similar solution, while O-MaSE gives a MAS architecture similar to ours, with personal agents to support the stakeholders, but centralised review assignment and paper selection.

If the system to develop is adapted to an AMAS approach (as verified in the first steps of *ADELFE*), such as this example, the proposed combined approach promotes the development of decentralised, distributed MAS for problem solving, and gives the possibility to deal with self-organisation of the collaboration links between agent instances, at a class (agent or role) level, which is not clearly representable in *TROPOS*.

The application of this approach combining the two modelling paradigms and metamodels is therefore restricted to a particular subset of systems, but provides higher expressivity; modelling also gains from the detailed guidelines available in *ADELFE* to identify system entities and agents and to define inter-agent cooperation.

However, there are different drawbacks. The emergent behaviour coming from the bottom-up approach to self-organisation, performed by modelling the single reactions to non-cooperative situations, can be validated only by empirical study, which is out of scope of this paper. Thus, the link between this bottom-up approach and the objective of the system is still not straight-forward. Still, we think that by combination with the top-

down *TROPOS* goal analysis and decomposition, we are able to shrink the gap between global system goals and cooperation rules.

A verification of such systems, particularly a verification of the emergent behaviour arising from cooperation rules, can only be achieved by testing. The approach proposed by Nguyen et al [14] derives *testing goals* from *TROPOS* goals and generates test scenarios by an automated, evolutionary technique.

5 Related Work

Currently, the works on methodologies focusing on self-organisation in multi-agent systems tends to increase. Tom de Wolf and Tom Holvoet have proposed a full lifecycle methodology customising the Unified Process [19]. At the requirement analysis phase, identification of macroscopic properties which must be shown by the running system is added to the classical steps. Then, the design phase is customised with two steps: one for deciding whether or not it is relevant to use a self-organising system and the other for exploiting existing practices and experiments. At the verification and testing phase, an empirical approach based on iterative development feedback is proposed. This method does not provide tools in order to choose existing approach to code self-organisation. The interesting and original part of this method is that it focuses on the system validation.

In [4] the authors present a case study of a decentralised multi-agent system for ambient intelligent scenarios, motivating the need of novel organizational structures of agents that result more flexible than traditional ones, e.g. broker and matchmaker, in order to deal with context changes. The architectural design phase has been conducted by the *TROPOS* modelling language in order to include the social surroundings needed to better characterize MAS architectural requirements. The resulting structure, *Implicit Organisation*, includes the self-organising property for the reassignment of the mediator role, i.e., the architectural requirement of disintermediation. Nevertheless, [4] does not detail the agent coordination level.

Gerhenson [11] proposes a domain-independent methodology for designing and controlling self-organizing systems. This iterative and incremental methodology includes several steps: Representation, Modelling, Simulation, Application and Evaluation, which are interrelated. The main point of this method is that the distributed control is also specified in order to always influence (by reducing friction and promoting synergy) the system and ensuring it will produce the desired behaviour. The work is more a philosophical work aiming at understanding these complex systems but also at designing them.

Another example of framework for engineering self-adaptive and self-organising systems is MetaSelf [8] which takes into account the design and the run-time levels. At the analysis phase, the requirements related to the properties of the components of the studied systems, the rules (local and global) that guide their behaviour and how the process development has to be carried out are identified. Depending on these properties, the relevant self-* architectural patterns and adaptation mechanisms are selected during the design phase. During the implementation phase, the run-time infrastructure, components, policies and metadata are developed. These frameworks rely on established

general principles that fit any kind of self-* system but some guides for developers are missing.

Gardelli [9] presents an approach to engineer self-organising MAS from the early design phases. The architectural pattern adopted is based on the Agents and Artefacts metamodel. Designing a self-organising MAS consists in embedding the self-organisation mechanisms in environmental agents and properly designing their interactions with the artefacts of the environment. The design approach comprises three-steps. Modelling first provides an abstract model of the system in which user agents, artefacts and environmental agents are characterised. The second step uses stochastic simulation to study the system dynamics through statistical analysis of results, considering that proper parameters are provided for artefacts and agents. The last step consists in tuning them until the desired dynamics appears. This proposal is mainly a guide for early-designing systems based on self-organising patterns that already exist such as natural ones.

6 Conclusion and Future Work

To promote the development of a decentralized, collaborative MAS, this work proposes to enhance the goal-driven AOSE methodology *TROPOS* with concepts and modelling steps from *ADELFE* methodology. The synergy of both software engineering methodologies allows the characterisation of a decentralised MAS by the definition of intra-agent coordination properties and enhances the expressivity of the *TROPOS* modelling language. The designer is now guided along a top-down goal-oriented modelling process analysing the intentions of the system's stakeholders, which is enhanced with specific design steps devoted to the bottom-up specification of agent coordination. The resulting agents are able to rearrange their cooperations, leading the MAS to optimise the achievement of its current organisational goal, bringing forth global emergent behaviour.

Traceability of requirements through the design phases until the definition of the agents behaviour is improved, reducing the conceptual gap by maintaining the concept of goal until detailed design and –if a BDI platform is used for the implementation, e.g. by a mapping as in [16]– even until run-time. This traceability is important especially if requirements change during system development and maintenance.

Future work concerns detailing the complete modelling process, leading to a goal-oriented methodology where the requirements analysis for adaptive systems can be conducted both at agent level and at organisation level. Besides, we want to further investigate on the benefits of including *TROPOS* goal modelling steps in the *ADELFE* methodology in order to improve the traceability of requirements through the design phases.

References

1. C. Bernon, V. Camps, M.-P. Gleizes, and G. Picard. Engineering Adaptive Multi-Agent Systems: The ADELFE Methodology. In B. Henderson-Sellers and P. Giorgini, editors, *Agent-Oriented Methodologies*, pages 172–202. Idea Group, NY, USA, June 2005.

2. C. Bernon, M. Gleizes, S. Peyruqueou, and G. Picard. ADELFE, a Methodology for Adaptive Multi-Agent Systems Engineering. In *Third International Workshop Engineering Societies in the Agents World (ESAW-2002)*, 2002.
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. P. Bresciani, L. Penserini, P. Busetta, and T. Kuflik. Agent Patterns for Ambient Intelligence. In *23th International Conference on Conceptual Modeling (ER 2004)*, LNCS 3288, pages 682 – 695, Shanghai, China, 2004. Springer-Verlag.
5. D. Capera, J.-P. Georgé, M.-P. Gleizes, and P. Glize. The AMAS Theory for Complex Problem Solving Based on Self-organizing Cooperative Agents . In *TAPOCS 2003 at WETICE 2003, Linz, Austria, 09/06/03-11/06/03*. IEEE CS, June 2003.
6. S. A. DeLoach. Modeling organizational rules in the multi-agent systems engineering methodology. In *Canadian Conference on AI*, pages 1–15, 2002.
7. S. A. DeLoach, L. Padgham, A. Perini, A. Susi, and J. Thangarajah. Using three aose toolkits to develop a sample design. *Int. J. Agent-Oriented Softw. Eng.*, 3(4):416–476, 2009.
8. R. Frei, G. D. M. Serugendo, and J. Barata. Designing self-organization for evolvable assembly systems. In S. A. Brueckner, P. Robertson, and U. Bellur, editors, *SASO*, pages 97–106. IEEE Computer Society, 2008.
9. L. Gardelli, M. Viroli, M. Casadei, and A. Omicini. Designing self-organising environments with agents and artefacts: a simulation-driven approach. *IJAOSE*, 2(2):171–195, 2008.
10. J.-P. Georgé, B. Edmonds, and P. Glize. Making self-organising adaptive multiagent systems work. In F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, pages 319–338. Kluwer Academic Publishers, 2004.
11. C. Gershenson. A general methodology for designing self-organizing systems. *CoRR*, abs/nlin/0505009, 2005.
12. M. Morandini, L. Penserini, and A. Perini. Automated mapping from goal models to self-adaptive systems. In *Demo session at the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008)*, pages 485–486, September 2008.
13. M. Morandini, L. Penserini, and A. Perini. Towards goal-oriented development of self-adaptive systems. In *SEAMS'08: Workshop on software engineering for adaptive and self-managing systems, Leipzig, Germany*, pages 9–16, New York, 2008. ACM.
14. C. D. Nguyen, S. Miles, A. Perini, P. Tonella, M. Harman, and M. Luck. Evolutionary testing of autonomous software agents. In *The Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pages 521–528. IFAAMAS, 2009.
15. A. Omicini, A. Ricci, and M. Viroli. *Agens Faber*: Toward a theory of artefacts for MAS. *Electr. Notes Theor. Comput. Sci.*, 150(3):21–36, 2006.
16. L. Penserini, A. Perini, A. Susi, M. Morandini, and J. 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.
17. L. Penserini, A. Perini, A. Susi, and J. Mylopoulos. High variability design for software agents: Extending tropos. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 2(4), 2007.
18. G. Picard, C. Bernon, and M.-P. Gleizes. ETTO: Emergent Timetabling Organization. In *International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS), Budapest, Hungary*, pages 440–449. Springer, September 2005.
19. T. D. Wolf and T. Holvoet. Towards a methodology for engineering self-organising emergent systems. In H. Czap, R. Unland, C. Branki, and H. Tianfield, editors, *SOAS*, volume 135 of *Frontiers in Artificial Intelligence and Applications*, pages 18–34. IOS Press, 2005.