Requirements Eng (2017) 22:77–103
DOI 10.1007/s00766-015-0236-0

ORIGINAL ARTICLE

Engineering requirements for adaptive systems

Mirko Morandini[1] · Loris Penserini[1] · Anna Perini[1] · Alessandro Marchetto[1]

Received: 15 October 2014 / Accepted: 7 August 2015 / Published online: 27 August 2015

Abstract The increasing demand for complex and distributed software calls for novel software engineering methods and techniques, to create systems able to autonomously adapt to dynamically changing situations. In this paper, we present a framework for engineering requirements for adaptive software systems. The approach, called Tropos4AS, combines goal-oriented concepts and high-variability design methods. The Tropos4AS requirements model can be directly mapped to software prototypes with an agent-oriented architecture which can be executed for requirements validation and refinement. We give a comprehensive description of the framework, with conceptual models, modelling guidelines, and supporting tools. The applicability of the framework to requirements validation and refinement is illustrated through a case study. Two controlled experiments with subjects provide an empirical evaluation of the proposed modelling language, with statistical evidence of the effectiveness of the modelling approach for gathering requirements of adaptive systems.

Keywords Requirements engineering · Agent-oriented software engineering · Adaptive software · BDI agents · Empirical study

✉ Mirko Morandini
mirkofbk@gmail.com

Loris Penserini
elpense@gmail.com

Anna Perini
perini@fbk.eu

Alessandro Marchetto
alex.marchetto@gmail.com

[1] FBK Center for Communication and Information Technology, 38123 Povo, Trento, TN, Italy

1 Introduction

Today's software is expected to be able to deal autonomously with dynamic and unforeseen changes, meeting users' needs and performance qualities, and avoiding failure. For example, a patient monitoring application for elderly persons in an assisted living residence is supposed to ensure the patient having regular meals and medicine without annoying him, while he moves from one (physical) context to another, or while he interacts with other people in unplanned ways.

Adaptive systems are proposed as a solution to realise such software applications, with the following key features: (1) they are aware of the objectives they should achieve; (2) they have a model of the environment, and are able to monitor it; (3) they are able to recognise when they deviate from their objectives and avoid failure; and (4) they are able to explore and select alternative behaviours.

Over the last decade, research agendas in software engineering were formulated based on an analysis of the new needs emerging for engineering such complex systems [20, 22, 56, 57]. Challenges were pointed out, including: how to deal with missing or uncertain information about the operational environment at the time the system is under design; how to make requirements available to the system at run-time (as a component of the system or of the platform on which the system is deployed), with the purpose of optimising requirements satisfaction while respecting quality of service, and how to synchronise the requirements changes with the software architecture. Focusing on requirements engineering (RE), it turns out that requirements analysis for adaptive systems is inherently incomplete [12]. Taking into account the variability in requirements for adaptive systems and the alternative design solutions resulting thereof, many decisions that are

traditionally made at requirements- and design-time need to be shifted to run-time [22, 72].

In our view to engineer adaptive systems, adopting a model-based approach, functional and non-functional requirements need to be analysed upon modelling:

- domain knowledge relevant to detail the goals the system is expected to satisfy,
- system capabilities as sets of variants that can be explored and selected at run-time to realise a desired behaviour in a specific context, and
- knowledge on how the system should act to avoid failure.

Since in many domains it would be impossible or hard to collect such knowledge automatically at run-time (e.g. based on machine learning techniques), the requirements analyst needs to give "hints" to the system and to "guide" it in properly interpreting contextual information in order to decide about *when* to change its behaviour and *which* alternative behaviour to select. At run-time, a monitor-analyse-plan-execute loop [30] can realise the adaptation by monitoring requirements satisfaction and making effective changes based on the knowledge modelled at requirements-time.

We developed a framework for engineering adaptive systems, called *Tropos for Adaptive Systems* (*Tropos4AS*) that provides analysts with modelling features for strengthening the requirements analysis on the specific knowledge and decision criteria needed by an adaptive system to autonomously adapt to dynamic changes.

We base the framework on three key modelling aspects, with the aim of capturing the specific requirements knowledge, and at the same time to reduce the gap between requirements-time abstractions and a run-time representation of requirements for adaptation. They are: (1) a *goal model* including information on goal types and associated satisfaction conditions; (2) an *environment model* representing those elements surrounding the system in different situations, and how they can affect the satisfaction of system goals; and (3) a *failure model* to support engineers to indicate unwanted states of affair and to allow them designing recovery procedures, either to anticipate foreseeable failures or to recover from unpredictable failures.

Tropos4AS also includes a mapping to software agents for prototyping and performing requirements simulation, thus supporting analysts during requirements validation and refinement. It thus aims at being useful also for analysts that are not experts in the modelling of goal-oriented and agent-based systems.

Different paradigms fit together in the *Tropos4AS* framework. First, goal-oriented requirements engineering approaches which aim at capturing stakeholder goals, with strategic dependencies for goal achievement, as introduced by *i** [70], and which exploit human-oriented abstractions such as *agents* and *goals* to model them and support their analysis to motivate software system requirements. Second, agent-oriented software engineering, which aims at operationalising (goal-oriented) requirements into an agent-oriented system design, such as within the software engineering methodology *Tropos* [9, 49]. Third, goal-directed software agents [8], providing suitable language constructs for requirements representation and reasoning at run-time, enabling the traceability of choices and failures back and forth between requirements- and run-time.

Our work complements research efforts conducted in the last ten years, which were also analysed in a recent systematic literature review [69] in the light of the modelling paradigms they use, thematic gaps left, as well as maturity level of the proposed requirements engineering methodologies. Indeed, goal-oriented modelling has been used in several approaches for engineering adaptive systems, while a synchronisation of the requirements to the system architecture, and an empirical evaluation of the effectiveness of the proposed modelling approaches, are among the poorly investigated topics that we aim at addressing in our work.

Contribution and Outline In this paper, we give a comprehensive description of *Tropos4AS*, providing conceptual models, a graphical language, and its semantics, to enable capturing requirements needed for defining and driving adaptation. We then give an overview on supporting tools and the possibility of requirements simulation. The paper elaborates and extends previous publications where preliminary fragments of the framework have been presented [41–43], by completing and detailing the concepts of the modelling language, defining meta-models and the modelling process and linking the models with the semantics defined in [44]. The applicability of prototype simulation during requirements validation and refinement is illustrated through a case study on engineering the requirements for the controller of a cleaner robot (called *iCleaner* from now on). Moreover, we present a thorough empirical evaluation of the proposed modelling language for comprehensibility, effectiveness in capturing requirements, and modelling effort. This is achieved by running controlled experiments with subjects for evaluating the use of *Tropos4AS* for requirements modelling, in comparison with the *Tropos* language.

The paper is organised as follows. Section 2 presents the *Tropos4AS* modelling language, with concepts, meta-models and the design process. Section 3 defines goal model semantics for representing requirements at run-time, and introduces the supporting modelling and code generation tools. Section 4 describes how *Tropos4AS* requirements models can be directly mapped to executable code, thus enabling prototypes simulation. For illustrating how *Tropos4AS* can give support for requirements validation

and refinement, a case study is presented. In Sect. 5, we detail an empirical study for the evaluation of the *Tropos4AS* modelling language in comparison with *Tropos* and discuss the results. Section 6 presents related work, while conclusions and future work directions are given in Sect. 7.

2 The Tropos4AS language

The *Tropos4AS* modelling language has been defined taking into account the key properties that need to be engineered into an adaptive system. Consider, for instance, the patient monitoring application (PMA) that needs to be aware of its own goals, which can be of different types and expressed at different levels of detail, as for example "follow medical prescriptions" and "assure the patient takes medicine before 8PM", the former being a state to be maintained in time (*maintain-goal*), the latter to be achieved at a certain time (*achieve-goal*).

Also, to make the PMA able to prevent failures such as an *inappropriate medicine supply*, patient monitoring functions need to be designed. An excerpt of the *Tropos4AS* model for the PMA is depicted in Fig. 15 left, which shows one of the possible monitorable errors, medicines not taken autonomously. If this error is not handled, it may bring to a severe system failure, e.g. medicines not taken. This failure can be prevented by specifying appropriate capabilities, e.g. call assistant. The system could need to adapt also autonomously, by using a novel combination of existing capabilities. For this, it is necessary to give the system awareness on the goals to reach, to define the effect of its activities on the environment and to give the possibility to monitor the environment for detecting deviation prior to goal failure.

2.1 Concepts and models

The *Tropos4AS* modelling language borrows concepts from *i** [71] and bases its development process on the agent-oriented software engineering methodology *Tropos*, as defined in [49], focusing on the actors representing the system-to-be.

It allows modelling requirements by considering how the system should behave to satisfy its goals. The three main dimensions included in the model are:

- an extended **goal model** and corresponding run-time satisfaction criteria,
- an **environment model** to represent the key elements in the context of the system, which affect goal satisfaction,
- a *failure model* to assist the requirements engineers in eliciting undesirable states of affair and possible recovery procedures either to anticipate foreseeable failures or to recover from unpredictable ones.

In goal-oriented RE [62, 71], goal models capture the stakeholder's objectives and define a system's high-level functional and non-functional goals and alternatives. *Tropos4AS* relies on goal models and the modelling process defined in the *Tropos* language [49]. Following the *Tropos* goal modelling process, goals are delegated from stakeholders to the *system actor*, which can be decomposed to one or more subsystem actors called *Agents*. Performing *goal analysis*, goals are decomposed (in AND and OR), re-delegated to other (sub)actors or operationalised by defining *plans*.

The corresponding process, depicted in Fig. 5, fits within the *Tropos* requirements engineering process defined in [38]. Each step of *extended goal modelling* and *failure modelling* may be performed iteratively or in parallel, though for clarity reasons only high-level iterations are shown in the figure.

2.2 Extended goal model

Tropos4AS adds details about the dynamics of goal satisfaction to goal models, a central aspect for enabling the monitoring of goal satisfaction and the decision-making in an adaptive system at run-time.

2.2.1 Goal types

Goals denote a state of affairs to be achieved [9]. *Tropos4AS* captures the dynamics of goal satisfaction, which can depend on other goals' (non-)satisfaction or on environmental circumstances, which may change also during the satisfaction process.

Following the unifying framework defined by Riemsdijk et al. [64], we endow goals with a basic goal life-cycle with the three states *adopted*, *active* and *suspended* (Fig. 1). Once the system needs to bring about a goal, the goal is *adopted*. An adopted goal can be either *active* or *suspended*. The agent needs to take some action if a goal is in the *active* state, but currently not *satisfied*. Upon this model, we instantiate three types of goals, also adopted in software agent programming [19], to define the attitude of an agent towards goal satisfaction.

Goal satisfaction can be necessary either for a period of time (*maintain-goal*) or once in a certain situation (*achieve-goal*). Furthermore, for practical reasons we include the possibility to process specific requests without reaching particular achievement conditions (*perform-goals*).

Achieve-goals are characterised by an achievement condition that specifies when a certain state of affairs is

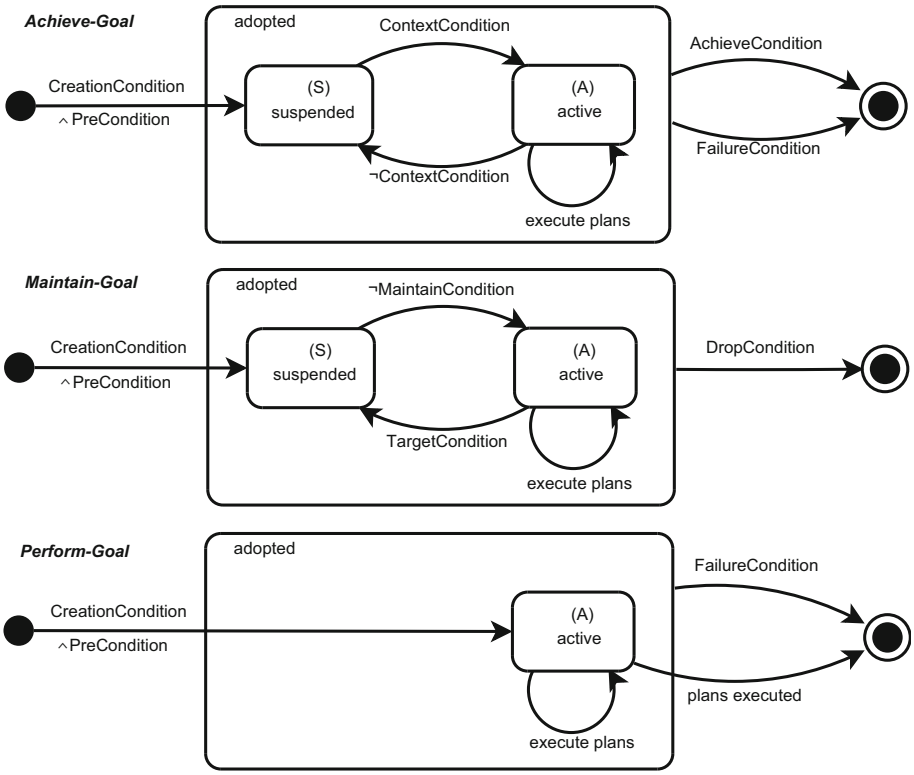

Fig. 1 State diagrams for the goal satisfaction process, adapted for the three basic goal types achieve-goal (*top*), maintain-goal (*centre*) and perform-goal (*bottom*), with associated conditions

reached. Once reached, these goals are dropped from the set of goals to be pursued by a system (i.e. the *active* goals).

Maintain-goals denote the need to maintain a certain state of affairs for a period of time. Different semantics can be attributed, such as a reactive or proactive satisfaction of goals [23].[1] Aiming at a practical implementation, *Tropos4AS* defines formal semantics for *reactive* maintain-goals, which call for action at the moment that a maintenance condition is not satisfied (cf. [44, 64]), and are suspended when the target condition is achieved. For instance, "follow medical prescriptions" is a maintain-goal in the PMA example, while "take medicines before 8PM" is an achieve-goal.

Perform-goals are satisfied by successfully executing one of the associated activities (i.e. plans or subgoals).

It is worth to notice that different goal types were also adopted in the *Formal Tropos* language [25]. However, the temporal logic-based semantics were defined for consistency verification applying model-checking techniques and thus defined semantics that are not suitable for a run-time application with imperative programming languages.

[1] Proactive maintain-goals require predictive reasoning mechanisms, and thus they are not easily representable in procedural agent languages in general [64], and approaches such as a "look-ahead with rollback" [28] are deployable only in specific domains.

2.2.2 Goal relationships

To express run-time dependencies between goal instances in a, possibly parallel, goal satisfaction workflow, *Tropos4AS* introduces a goal *sequence* relationship and the *Inhibition* relationship. *Sequence* denotes a sequential order for the adoption and achievement of two goals. *Inhibition* expresses run-time precedence between goals in the following form: if a goal *A* inhibits a goal *B*, as long as *A* is in an *active* state, the achievement process of *B* has to be suspended. For example, in a cleaner robot, a goal recharge battery needs to *inhibit* goals related to cleaning.

2.2.3 Softgoals

Softgoals are used to express user preferences and QoS needs. An *importance* value is added to *Tropos* softgoals, denoting the current importance of a softgoal for an actor in a range [0,1]. Changing importance at run-time, a software can adapt to changing quality needs by satisfying alternatives which were previously not considered. Change of importance could be manual (e.g. defining user preferences in a personal agent), or by using supervised learning approaches [31].

2.3 The environment model

Adaptivity is concerned with *how* a software system behaves when both expected and unexpected changes

occur in its environment. With the environment model, the dependencies between the agent's goals (which determine the agent's behaviour) and its intentional and non-intentional environment [35] are made explicit.

2.3.1 Intentional actor relationships

For representing dependencies between intentional actors in goal models, we use *Tropos* (*i**) strategic dependency modelling. A goal dependency between agents expresses a delegation of goal achievement or maintenance, a plan dependency represents a delegation of performing some activity, while a resource dependency expresses the need for obtaining data. Following the *Tropos* process, goals, plans and resources delegated from a depender actor need to be reported and detailed in the dependee actor. An implicit creation condition for delegated goals and plans is given by the depender's goal which needs the dependency. Also, temporal relationships (e.g. precedence) can be expressed implicitly, by modelling dependencies to resources that need to be produced before being consumed.

At an actor level, adaptivity can also concern the selection of agent instances and the selection of agents within the ones covering a certain role. This type of adaptivity is not covered by *Tropos4AS*. For capturing adaptivity of dependencies at instance-level, an approach like the one proposed in [40] can be applied.

2.3.2 Non-intentional entity relationships

For relating changed circumstances to appropriate system behaviours, we define non-intentional entities following the definition of *Artefact* [47]. As opposed to actors or agents, artefacts are (as perceived from an agent's perspective) entities without an autonomous, proactive behaviour, used by agents to sense or act in the environment or to hold data for interchange, providing *functionalities* through usage interfaces, and properties describing their *inspectable* internal state.

Figure 2 depicts a fragment of the *Tropos4AS* meta-model focusing on the environment. An agent (representing a stakeholder or the system) is composed of a goal model, the agent's knowledge represented by beliefs related to data sensed by artefacts and the agent's own state at runtime, including the goal satisfaction life-cycle and its achievement state. An *artefact* in the environment can either be *internal* or *external* to the system, to distinguish artefacts in the boundary that is currently modelled and implemented, from external, given ones, whose properties cannot be influenced. It is important considering that the environment can also be a medium for sharing information and mediating coordination among systems [66].

We model the agent's view on the system environment by eliciting the artefacts the system affects or is directly affected by, and by analysing the relationships between the system's goals and these artefacts. This is done iteratively and in parallel with goal conditions modelling, presented on the following pages. Primary candidates are *resources* modelled in *Tropos Late Requirements Analysis* goal diagrams, if they define artefacts used to sense or act in the world, and the services and interfaces inside and outside the system boundary, which need to be accessed. If available, domain models or ontologies can help, either to identify artefacts or to represent the environment model itself. For instance, in the case of an ambient assistance application such as our example PMA installed in a social residence, the environment is represented by the various sensors available and the various entities they can track, including the patient. For the controller of a cleaner robot, as stylised in Fig. 6, the environment would contain the battery (internal to the system) and the room with its obstacles and the dustbin (external).

The environment model is kept simple deliberately, to focus on the relationships between environmental entities and the agent. It does not aim to model the whole domain including relationships between entities, or to detail their internals. Available modelling languages, from UML class

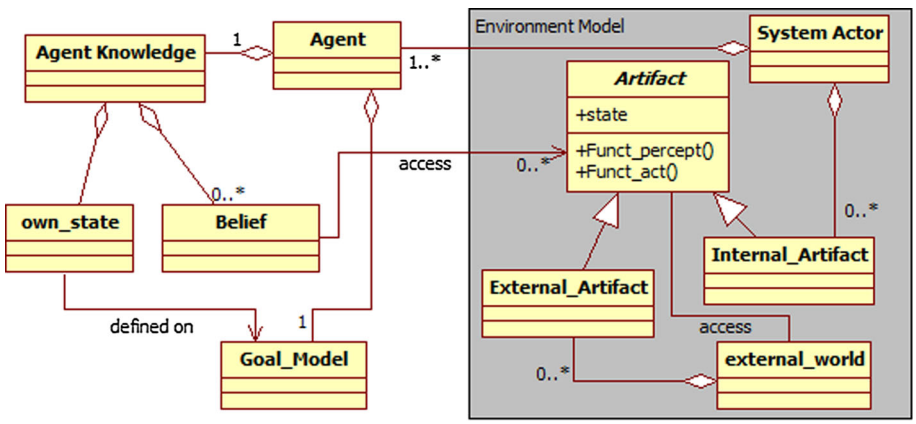

Fig. 2 Detail of the *Tropos4AS* meta-model, focusing on the relationships of the agent's knowledge with the surrounding environment, represented by artefacts

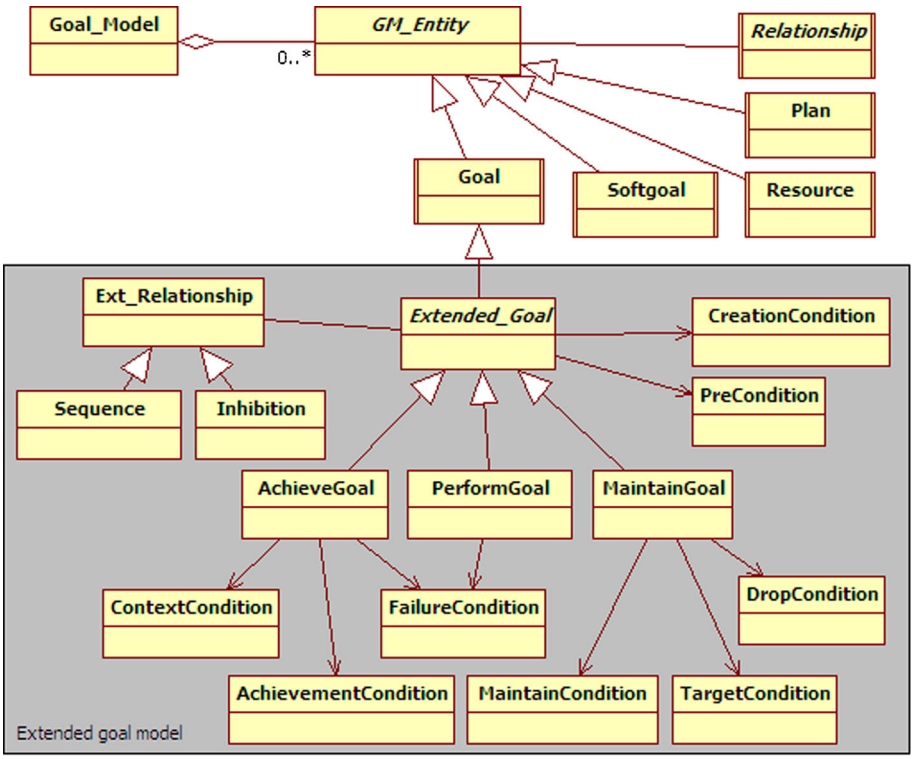

Fig. 3 View on the *Tropos4AS* meta-model showing the goal model with the extended goal concept, including goal types and conditions

diagrams to domain ontologies, can be used for this purpose, depending on the needs and the extent of a project.

2.3.3 Goal conditions on environmental states

To define the run-time goal satisfaction process in response to environmental changes, *Tropos4AS* uses the concept of *condition*, a choice supported also by various agent programming languages (Fig. 3).

Specific **conditions** are associated with each goal type, to define the run-time goal satisfaction process in response to changes in the agent's view on the environment, either guiding state transitions in the goal satisfaction process (i.e. satisfaction triggers a state change) or guarding them (i.e. satisfaction is a prerequisite for a state change). Conditions are boolean expressions evaluated on the state of the own knowledge, which holds the state of own goals and the own belief on artefacts in the environment (see Fig. 4), represented by environmental artefacts. For example, a value reported by a sensor on the medicine dispenser managed by our PMA can be related to the achievement of a goal take medicines before 8PM by a condition such as *num-medicines-taken* $\leq$ *num-mandatory-medicines*. These expressions can be defined, depending on the purpose and the granularity of the model, in natural language, or as an expression on properties and functionalities delivered by the artefacts.

Figure 1 illustrates how the state transitions for the three goal types are guided or guarded by the satisfaction of the different conditions:

Creation conditions (CC) determine the criteria for adopting a goal and thus starting the goal satisfaction process, transiting, depending on the goal type, from the *non-adopted* to the *suspended* or *active* state.
Preconditions are guarding conditions that have to be fulfilled to adopt a goal, and are modelled to restrict the adoption of a goal to a particular context.
Context conditions are used to suspend goal achievement. They have to be valid during goal achievement. A goal in *suspended* state transits to the *active* state if the condition is satisfied and vice versa.
Achievement conditions define when an *adopted* achieve-goal was satisfied and will thus be dropped. Note that the name (i.e. the label) of a goal is typically a (often vague or ambiguous) description of its achievement condition.
Maintain and *target conditions* characterise start and end of the maintain-goal satisfaction process—maintain conditions activate the process while target conditions suspend it;
Drop conditions denote a state in which it is no more necessary or desired to try goal satisfaction (especially for maintain-goals), typically defining a temporal or environmental limit.

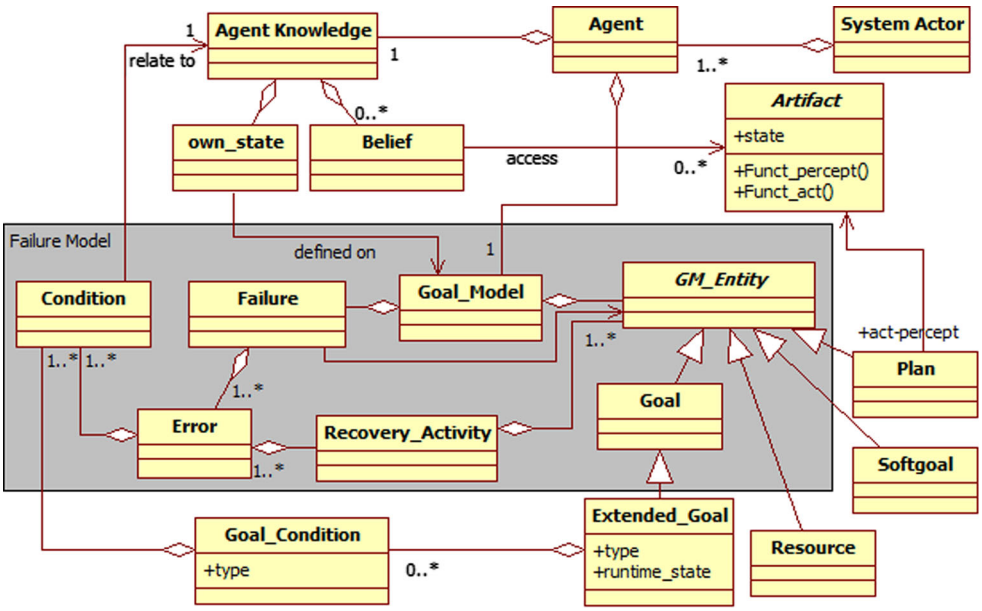

Fig. 4 The Tropos4AS meta-model with details on condition and failure modelling (goal model simplified)

Failure conditions Failure conditions denote states in which it would be impossible to achieve a goal; the satisfaction of such conditions thus denotes the failure of the goal. While failure of a root goal is critical and will lead to system failure or at best to a system working in a degraded mode [3], failure of subgoals can be caught at a parent goal level, e.g. by exploring modelled alternatives.

In the *Tropos4AS* modelling process (Fig. 5), making these conditions explicit, new monitoring requirements may be identified, leading to new or changed environmental artefacts. The environment model needs thus to be revised in parallel. Also, goal types (*maintain*, *achieve*, *perform*) may be defined or refined in parallel, in compliance with the meta-model (Fig. 1). The purpose of extended goal modelling and condition modelling is, however, not to define goal models formally and completely, but to capture and specify the requirements and the known information about what an agent has to observe and consider for goal achievement. This aims at enriching the agent's context awareness.

The formal semantics of the expected behaviour exhibited by the agent for the satisfaction of the three types of goal in a goal model, along with the different conditions that determine their life-cycle, are briefly described in Sect. 3 and detailed in [44].

2.4 The failure model

Variability modelled in a goal tree through *OR*- and *Means-end* decomposition [49] is a source for adaptivity, but a goal model typically describes scenarios which follow a "happy path", i.e. scenarios defining the default software behaviour when all works as intended.

Tropos4AS introduces a failure model, to provide a means for capturing exceptional situations and possible mitigation activities at requirements-time, to anticipate failure at run-time. A failure of the software system is mostly caused by an incompleteness of the requirements, namely, by circumstances (i.e. inputs) that were not considered at requirements analysis and design-time. For example, the physical failure of an indispensable part of a robot does not imply the failure of the control software, as long as this failure was anticipated and correctly handled. The failure model gives a direct and graphical means to the analyst to elicit possible errors that cause failure and to analyse the possibilities to recover from these errors.

The following concepts are introduced (Fig. 4) to capture failures at a requirements level: Goal **Failure** situations are states of the world that explicitly capture the inability to satisfy a goal. **Errors** are perceivable events (defined in the form of boolean conditions) that may cause failure. **Recovery Activities** provide new capabilities or a new arrangement of known capabilities to cope with a particular situation, to *anticipate failure*, *preventing* or *recovering from* an error. The recovery activities are also represented as (fractions of) goal models. For example, in the PMA, *inappropriate medicine supply* is considered a failure. One of the possible monitorable errors is medicines not taken autonomously, which would lead to a preventive recovery activity call assistant (see also Fig. 15).

2.4.1 Failure modelling

In the following, we describe the failure modelling process, consisting of failure identification, error elicitation, and recovery activity modelling, and goal model completion (Fig. 5b).

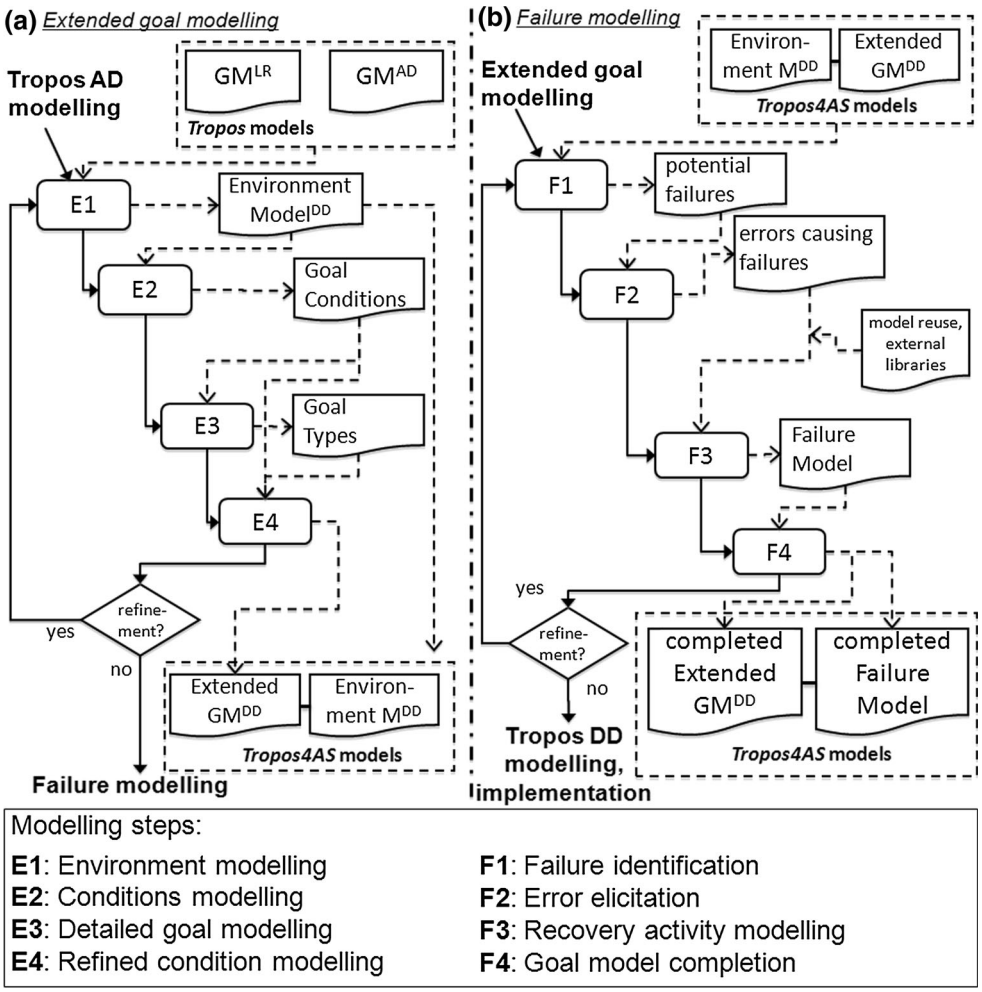

Fig. 5 Steps of **a** extended goal modelling and **b** failure modelling, including models in input and output of each step. Activities are depicted in *rectangles*, while *document-shaped elements* represent artefacts used or produced

Failure identification The requirements engineer analyses each goal of an agent for the possibility of failures in the achievement process, i.e. a set of states in which an agent (with its actual knowledge and with the given capabilities) is unable to reach a goal. Failures can be identified by analysing the domain of a goal, the modelled conditions, and the possible variability in the environment of the system. Identified failures (e.g. a failure battery discharged for a goal Battery loaded, see Fig. 6. A brief description of the failure in natural language is recommended. First candidates for associating a possible failure are goals with failure conditions or drop conditions, in the case that the resulting goal failure is not handled at a higher level (e.g. by available alternatives to achieve a parent goal). An analysis of failures may be of particular importance for goals that have yet no alternative, and goals without a parent goal that could mitigate their failure.

Error elicitation Errors can be discovered following a top-down process to find causes for a failure, or bottom-up, by analysing failures happening outside the system boundary that can affect goal satisfaction. They are defined in natural language and then related to the failure and the environmental artefacts concerned for monitoring the particular system state. For each error identified, it can be decided to not mitigate it because it involves only a reduced risk, or because mitigation would be more expensive or time demanding than human intervention in the case of system failure. To evaluate risks in detail, a separate risk analysis (e.g. [2, 11]) can be performed.

Recovery activity modelling Failures predictable by monitoring can be anticipated by implementing missing capabilities (e.g. for a cleaning robot, the ability to clean from a specific type of dirt) or by using a novel combination of existing capabilities, while failures not predictable by monitoring can only be mitigated at design-time, e.g. by making the system more robust, or by giving it the ability to detect errors that lead to the failure (e.g. by employing proper sensors), thus making the failure predictable. Recovery activities define ways for failure mitigation, considered already during requirements analysis, or as alternatives to be chosen at design- or at run-time, depending on the kind of failure and the possibility to

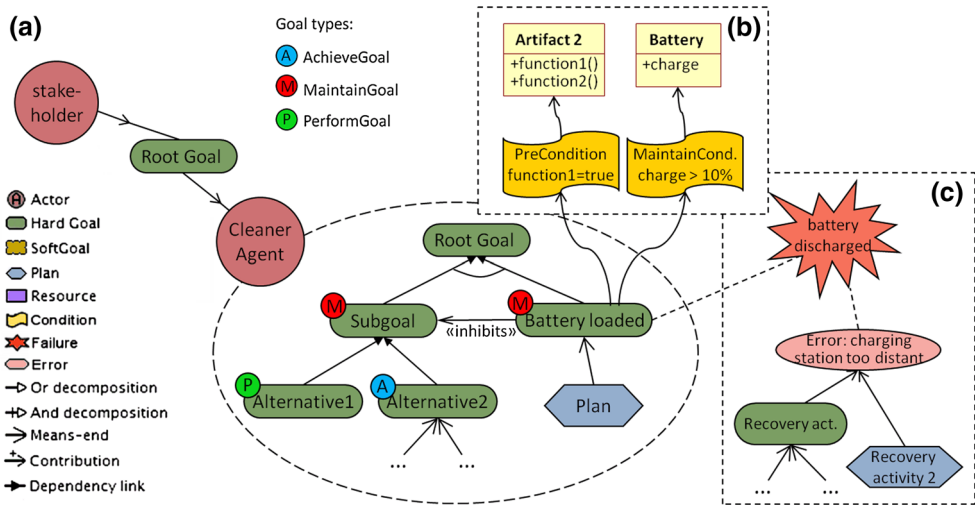

Fig. 6 Graphical representation of the modelling concepts introduced, illustrated on the model of a cleaner robot. **a** Extended goal model with goal types and new goal relationships; **b** environment model with conditions and artefacts; **c** failure model with failures, errors and recovery activities (which can also include parts of the original goal model)

anticipate it. Possible approaches are: (a) to anticipate goal failure, modelling new alternative capabilities executed in the specific error state; (b) to anticipate a failure of available capabilities (or of entire goal trees) by executing the recovery activity; (c) to try to reduce failure probability already at design-time, if a failure cannot be prevented, e.g. by making the hardware more reliable, or by restricting the operation environment.

Recovery activities are modelled following the standard *Tropos* plan modelling process [49], possibly reusing parts of the existing goal models. This conceptual notation allows to abstract from the complexity of techniques related to error diagnosis, creating intuitive and simple models that relate errors to recovery activities.

Goal model completion and refinement After modelling errors and recovery activities, we need to decide whether they should be kept either as an alternative within the core part of a goal model as part of the normal workflow of the system, or whether they cover an exceptional situation and should thus, according to the principle of separation of concerns, not be part of the agent's nominal behaviour (Activity F4 in Fig. 5). This choice is highly dependent on the current application domain.

If it is recognised (also e.g. by considering simulation results) that the potential causes for failure are recurrent events, or the execution of recovery activities is continuous or planned in advance, this should lead to the modelling within the agent's goal model, in the form of goals to achieve or maintain. On the other side, for rare events that bring about a deviation from an agent's main activities, like a failure in hardware/software or in intra-agent communication, it would be preferable to let the recovery activities in the failure model.

Being external to the goal model, the failure model can also possibly (depending on the selected implementation architecture) be updated at run-time with new failures, errors and recovery activities without interfering with the normal behaviour. Despite being outside the scope of this article, employing a learning mechanism that keeps track of successful application of capabilities would also be feasible with this approach. This would contribute to the realisation of a feedback mechanism, where engineers are informed of changes in the failure model and can consider them for integration in the subsequent development cycle, supporting an adaptive architecture as outlined in [53].

Several additional techniques available may be applied for refining and completing *Tropos4AS* goal models, borrowing ideas from the fields of obstacle resolution [63] and risk mitigation.

For identified failures, additionally to *recovery activity modelling*, a goal substitution or the definition of alternative goals, or a weakening (relaxing) of goals [67], can be explored to mitigate the failure from a requirements point of view. The modification of the decision criteria for selecting behaviours, and the iterative improvement of conditions, contributions and goal relationships needs also to be considered in the process.

Mitigation can be achieved also by improving the sensing of the environment, to timely detect the situations (errors) that lead to the failure (e.g. by employing special sensors for distance measurement, to notify an imminent crash). In this case, the failure becomes predictable and recovery activity modelling can be applied effectively. On the same side, as far as the system environment is accessible and modifiable, the requirements engineer can also try to enact changes in the environment to reduce the likelihood of failure.

Finally, also the dependencies to other agents need to be considered for enacting mitigation activities, either, by reduction of critical dependency to agents that are exposed

to failure, e.g. by implementing the needed capabilities locally, or, on the other side, by delegation of goals or tasks to other agents, which have proper knowledge and capabilities and can deliver a service with the requested qualities. In general, a detailed analysis of possible failures may be of particular importance for goals that have yet no alternative, e.g. goals in an AND-decomposition and, in particular, root goals, that are, goals directly delegated by the stakeholders, without any parent goal that could mitigate the failure.

2.5 Visual notation

The adopted *i**-based visual notation is illustrated in Fig. 6. Goal types are represented in a small circle at the upper left corner of a goal, while *inhibition* and *sequence* are represented by arrows labelled «inhibits» or «seq», respectively. The adopted graphical notation for the environment model is the UML class diagram, with artefacts represented as classes characterised by the functionalities they provide (as methods) and their state variables (as attributes), possibly grouped in packages, and can be detailed by using association, aggregation and generalisation relations.

Conditions are represented by flag-shaped boxes linked to a goal and one or more artefacts in the environment. Failures are represented by "jagged" circles, associated with the respective goal by a dashed line, while errors are represented by an ellipse. Recovery activities are modelled by goals and plans by using the *Tropos4AS* notation.

The modelling language largely complies with a wider effort for unifying the visual representation of goal modelling languages [48].

3 Goal model semantics

The concept of *goal* is a key element to concretise the influence and correlations between high-level requirements and system functionalities. By the use of goal types and their conditions, *Tropos4AS* allows to define a wide spectrum of agent's behaviours. In this section, we sketch an equivalent run-time semantics for *Tropos4AS* goal models that formally defines the process of goal satisfaction, hereafter called the *goal life-cycle*, and is the basis to build a formal model for the mapping process between requirements- and run-time concepts.

Tropos4AS introduces expressive extensions to goals: goal types and their conditions. Goal types precisely detail the agent's life-cycle by defining the run-time behaviour for achieving a goal. Conditions guide and guard state transitions in this life-cycle. Despite available agent-oriented programming platforms make large use of goal types and conditions, huge differences exist in available goal types, terminology and semantics [19]. These differences were addressed by a proposal for unification by Riemsdijk et al. [64]. However, since goal models (i.e. goal hierarchies) are not natively available in any of these languages, using goal types in goal models necessitates a formal definition of the behaviour intended at the time of modelling.

Different to other approaches where leaf goals fully define their parent goal [21, 25, 62] (where, e.g. an AND-decomposed goal is defined as the intersection between its subgoals' states), and it is only possible to state conditions for leaf level goals or tasks, we define also semantics for higher level goals, and thus the high-level decision-making process of an agent on the selection of (sub)goals to achieve.

The dynamic behaviour of *Tropos4AS* failures is defined by a mapping to goal models. Errors are mapped to achievement goals with the error condition as creation condition, while recovery activities are associated with them with a means-end relationship.

We formally bind the intended dynamic goal model semantics with a run-time behaviour by the definition of an operational semantics for goals in a hierarchical goal model, building upon the unifying semantics for goal types in [64]. The goal achievement run-time behaviour is defined through the achievement of its subgoals and the satisfaction of its achievement conditions.

At run-time, a goal will be at some time instantiated in the system (i.e. *adopted*). It can then be for some time in an *active* or *suspended* state, until being *dropped* because of success or failure. Several additional states and transitions have to be defined for goals in a hierarchical model. We adopt a minimal set as depicted in Fig. 7: *suspended* (S), *active_deliberation* (AD), *active_failed* (AF) and *active_succeeded* (AS).

We define an abstract architecture to capture a large pool of possible goal achievement behaviours, and instantiate it to formalise detailed semantics for achieve-goals, maintain-goals and perform-goals, in AND- and OR-decomposition. The instantiation is performed by linking goal conditions to a set of *transition actions* (one of ACTIVATE, SUSPEND, FAIL, SUCCEED, RETRY, REACTIVATE, DROPFAILURE, DROPSUCCESS).

The operational semantics are defined by a set of inference rules that define possible state transitions where each rule is specified as

$$\frac{L}{R} \qquad [\textit{rule-name}]$$

where R represents a possible state transition of the system under the set of conditions L. In this manuscript, we limit to an explanation of the prominent inference rules. An exhaustive definition for all possible state transitions, instantiated to various achieve-, maintain- and perform-goals, can be found in [37, 44].

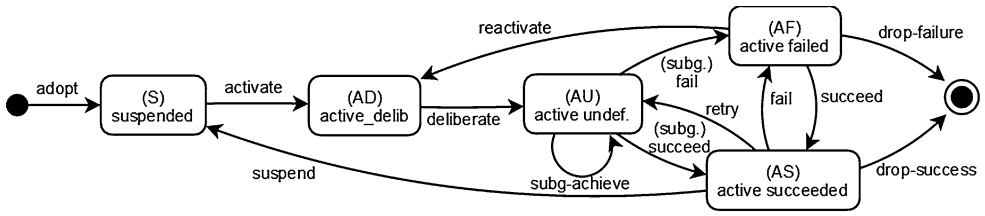

Fig. 7 Meta-model of the life-cycle states of a goal in a goal model, covering the three main goal types

The state of an agent is characterised by a tuple $\langle B, G\rangle$, where B is the set of actual beliefs and G is the actual set of goals $g_1 \ldots g_n$ the agent has to pursue (i.e. the adopted goals). We define a generic non-leaf goal at run-time as $g(C, E, s, \Gamma)$, where $s \in S$ is the actual goal state and Γ is a set of goals that results from a deliberation activity *deliberate*(g, B) that returns all the applicable subgoals γ_i. C and E are sets of tuples used as transition conditions, of the form $\langle condition, action\rangle$, where *action* is one of the transition actions and *condition* is evaluated in the agent's belief B. A condition c in C is evaluated in B if $\Gamma \neq \emptyset$ (i.e. the set of adopted subgoals is not empty), while a condition c in E is evaluated if $\Gamma = \emptyset$.

Suppose to have a goal g, OR-decomposed into 3 alternative subgoals γ_1, γ_2, and γ_3. According to the general state transitions defined, illustrated in Fig. 7, a goal gets adopted and activated by a creation condition. Then, in state *active-deliberate*, deliberation takes place: all subgoals that are applicable (i.e. they have no violated precondition with the agent's current belief of the environment) change to state *AU* (see Fig. 7). In this state, the agent will try to adopt one of the subgoals. Once a subgoal has crossed its own life-cycle states, depending on subgoal success or failure, one of the rules *[OR:subg-achieve]* or *[OR:subg-succeed]* is applicable:

$$\frac{\gamma_i \in \Gamma \langle B, adopt(G, \gamma_i)\rangle \rightarrow \langle B', G\rangle B' \models failure(\gamma_i)}{\langle B, g(E, AU, \Gamma)\rangle \rightarrow \langle B', g(E, AU, \Gamma \setminus \{\gamma_i\})\rangle} [OR : subg\text{-}achieve]$$

$$\frac{\gamma_i \in \Gamma \quad \langle B, adopt(G, \gamma_i)\rangle \rightarrow \langle B', G\rangle B' \models success(\gamma_i)}{\langle B, g(E, AU, \Gamma)\rangle \rightarrow \langle B', g(E, AS, \Gamma \setminus \{\gamma_i\})\rangle} [OR : subg\text{-}succeed]$$

The next transition rule defines how to satisfy the main precondition of these two rules, the transition from $\langle B, adopt(G, \gamma_i)\rangle$ to B', that is, adopting the subgoal γ_i in order to start its achievement process, and waiting until γ_i is dropped:

$$\frac{adopt(G, \gamma_i) \rightarrow G \cup \{\gamma_i\} \quad \langle B, G \cup \{\gamma_i\}\rangle \rightarrow \langle B', G\rangle}{\langle B, adopt(G, \gamma_i)\rangle \rightarrow \langle B', G\rangle}$$

The function *adopt*(G, g) performs the adoption of a subgoal, that is, adding the (sub)goal g to the current list of adopted (and not yet dropped) goals G in order to start its achievement process. Eventually, this will result in a new belief B'. This adoption function is currently performed by considering sequence and inhibition relationships and by prioritising alternatives by maximising the contribution of whole goal subtrees to softgoals, weighted by the softgoals' actual importance at run-time. The new belief B' is the result of the application of transitions for the satisfaction of the goal γ_i that concludes with some transition rule that drops γ_i from G.

Suppose that the first adopted subgoal succeeds. Thus, the rule *[OR:subg-succeed]* will be applied and the goal transits to the state *AS*. Depending on the instantiated goal type, now the goal will be directly dropped with success (transition *drop-success*) or the goal's success condition is evaluated. Once more, depending on the goal type definition, if the success condition is false, the goal may fail completely (transiting to *AF*) or be put back to the state *AU*, where another alternative subgoal will be adopted.

The following transition rule defines dropping of a goal in case of success, linked to a transition condition DropSuccess.

$$\frac{g(E, AS, \emptyset) \in G \quad \langle c, \text{DropSuccess}\rangle \in \text{E B} \models \text{c}}{\langle B, G\rangle \rightarrow \langle B \cup success(g), G \setminus \{g(E, AS, \emptyset)\}\rangle} [drop\text{-}successE]$$

The abstract architecture is instantiated by linking its transitions to the goal conditions of a specific goal type. As example, we report a simplified part of the formal definition for an achieve-goal $A(s, f)$ with success and failure conditions s and f, which is dropped with success when s is satisfied, dropped with failure when f is satisfied, and retried, otherwise.

$$A(s,f) \equiv g(\{\langle s, \text{DropSuccess}\rangle, \langle \neg s, \text{Retry}\rangle, \langle f, \text{Fail}\rangle, \langle f, \text{DropFailure}\rangle, \ldots\})$$

4 Tool support and illustrative case study

A key purpose of the *Tropos4AS* modelling approach is to lower the conceptual gap between requirements, design and implementation concepts. With this objective, for the implementation we propose to build on a BDI-based

architecture[2] that provides an explicit notion of goals, plans and goal satisfaction, and thus allows a homogeneous and coherent mapping between similar concepts available at requirements- and run-time. A mapping from Tropos4AS design concepts to structures of a programming language for BDI agents was defined, with the purpose of automating code generation, for obtaining executable prototypes that implement the requirements specification. It targets a rapid construction of prototypes of the behavioural model of a system.

4.1 Tool support and prototype generation

The mapping to agent code is supported by the visual modelling tool Taom4E.[3] An overview of the tool-supported *Tropos4AS* process is represented in Fig. 8. The Taom4E modeller implements the *Tropos4AS* meta-model and provides an editor for extended goal models and environment models.

From goal models, the code generation tool *t2x* generates executable code skeletons for the behavioural level of an agent. It relies on the *Jadex* Active Components platform,[4] which provides an explicit representation of active components (agents) and goals at run-time. In Jadex, goal models are mapped to Jadex goals (defined in XML) along with plans (Java) building the connection between them at different levels, with the help of a *Tropos4AS*-specific middleware that manages the decomposition logic for realising the semantics defined. In this way, the goal tree is resembled, with an execution behaviour according to the semantics defined. The approach is, however, mostly technology independent and can be applied similarly to other agent-based platforms.

An agent's goal model is coded in Jadex by mapping the goal decompositions to a Jadex structure composed of goals and associated plans for building the connection between goals at different levels in an *agent definition file* (ADF), see Fig. 11 for an example. A middleware manages the decomposition logic following the semantics defined in the previous section. This choice is motivated by the fact that Jadex, similar to other existing goal-oriented agent programming languages, such as Jadex, Jason and 2APL, do not support goal hierarchies as a native feature.

The goal decomposition graph is stored in the agent's belief base, together with all contributions to softgoals and dependencies to other agents. The resulting software is aware about its abilities, namely, at run-time it can monitor and control its behaviour by navigating the modelled goal graph, to select goals and plans according to the modelled requirements. Furthermore, code for resolving dependency links between agents is generated, using FIPA-standard request protocols. At run-time, agents are able to select between modelled alternatives (OR-decompositions, means-end relations, recovery activities), based on the evaluation of contributions to softgoals, of modelled conditions, and of subgoal failure.

```
<agent ...>
  <beliefs>
    <belief name="battery" class="battery">
      <fact>new Battery()</fact>
    </belief>
    <beliefset name="stations" class="LoadingStation">
      <fact>new LoadingStation(Position)</fact>
      <fact>new LoadingStation(Position2)</fact>
    </beliefset>
  </beliefs>
  ...
</agent>
```

Fig. 8 Excerpt of the belief base in an agent definition file

The agent's view on artefacts in the system environment is represented as facts in the agent's belief base and to Java classes providing an interface to the requested monitoring functionalities. For example, the battery of a cleaner robot maps to a Java class instantiated as a fact in the Jadex belief base, named battery (Fig. 9), and can be accessed from the ADF as well as from Java classes through an API.

Environment diagrams can be exported to UML class diagrams for the detailed design. Java code can be generated from the final UML models with available tools, such as Omondo EclipseUML or Papyrus. The implemented goal models are prepared to access them for the evaluation of modelled conditions.

The different types of conditions, defied in Sect. 2.2, are mapped to Jadex goal definition, using boolean formulas to link the goal achievement process to facts in the belief base, which represent the artefacts in the environment and are implemented in Java classes containing the methods that represent the functionalities of the artefacts. Goal sequence in an AND-decomposition is annotated in the ADF and implemented the Java plans defining the AND-goal decomposition logic, whereas the *inhibition link*, a concept available also in the Jadex language, can be directly mapped to an XML tag of the form `<inhibits ref="goal_to_inhibit"/>`.

Modelled failures are not directly represented in the agent code. Error conditions and the respective *recovery activities* are mapped to independent parts of the goal model structure within the agent definition. Error

[2] BDI (Belief–Desire–Intention) agents endow a reasoning cycle, are able to monitor the environment to update their belief, and to deliberate on the goals to achieve, selecting suitable plans [54].

[3] Tool for Agent-Oriented visual Modelling for Eclipse and its plugin *t2x* (*Tropos4AS* to Jadex), developed by the Software Engineering group at Fondazione Bruno Kessler (FBK), Trento [41, 50], available, including the extensions, at http://selab.fbk.eu/taom.

[4] http://www.activecomponents.org

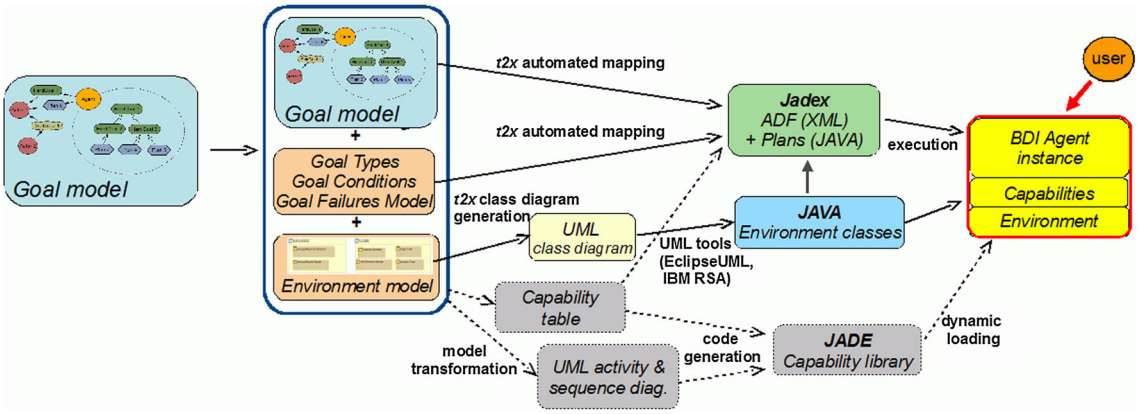

Fig. 9 Tool-supported Tropos4AS development activities and design artefacts, when going towards a goal-directed agent implementation in *Jadex*. *Dotted lines* complementary capability implementation approach, presented in [49]

conditions are implemented as creation conditions for the top-goals or plans of the goal models related to their *recovery activities*. In this way we obtain a behaviour such that, each time a specific error occurs, the goal model of one of the associated recovery activities is activated.

The tool is conceived for a fast prototyping of the behavioural model of a system.[5] The generated software agents are executable in Eclipse on the integrated Jadex platform. The resulting software provides basic adaptation mechanisms by exploring the modelled alternatives at runtime, optimising contributions to softgoals and mitigating modelled failures.

4.2 Prototyping for requirements validation and refinement: the iCleaner case study

The resulting software agents can be used to simulate the adaptive behaviour specified in *Tropos4AS* models, thus providing support for the validation of the modelled requirements.

Figure 10 depicts the requirements validation and refinement process that we instantiated with *Tropos4AS* to evaluate its applicability through a case study. Requirements modelling and analysis is performed using *Tropos4AS*. The resulting models are coded as JADEX agents, and in parallel test cases are derived with the *eCat* tool [45], following the goal-oriented testing methodology described in [46]. The *eCat* tool includes an automatic generation of test cases (new environments as well as dynamic changes in environments during a test) from goal models and from an ontology of the environment with an evolutionary, ontology-based algorithm, a framework for executing the test cases, and evaluation and reporting functionalities, to monitor the communication among

[5] It has been successfully used in university courses on agent-oriented modelling and programming at the university of Trento.

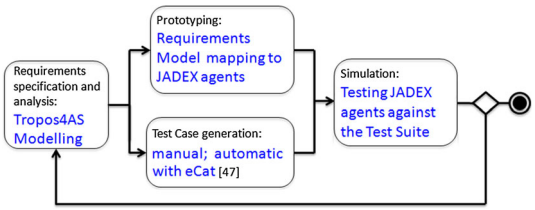

Fig. 10 Requirements validation and refinement through prototyping

agents and all events happening in the execution environment in order to trace the behaviour and to report errors.

During the simulation phase, the resulting test suites are executed to exercise the software agents and validate their behaviour. Visual animations of the specification can be obtained, by taking advantage of the graphical inspection tools provided by the Jadex platform. The feedback obtained by analysing the monitoring logs and observing the execution is then used to improve the requirements models, iterating the process till the intended objectives are achieved.

4.2.1 The iCleaner case study

We aim at modelling the requirements for the controller of a cleaning robot, **iCleaner**, which has to properly clean the floor of a room, by performing the following tasks autonomously:

1. Explore the area for important objects (waste and obstacles).
2. Collect waste and bring it to the closest bin which is not full.
3. Maintain the battery charged, by sufficient recharging.
4. Avoid obstacles, by changing course when necessary.

The cleaning robot needs basic adaptivity features to deal with the dynamic environments where attributes of

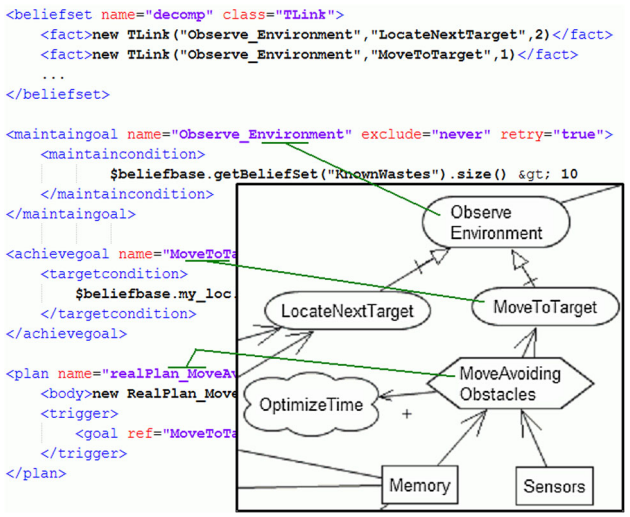

Fig. 11 Part of a goal model and the corresponding generated Jadex ADF

objects may change (e.g. locations of obstacles), or unknown objects may appear. Adaptivity allows the agent to keep or improve its performance as well as its robustness. The performance of the agent can be calculated based on its efficiency (the waste collected in a time slot) and its power consumption, while the robustness of the **iCleaner** is estimated by the number of crashes during a unit of operation time.

The case study was conducted by four researchers experienced in goal modelling and agent-oriented programming (including two authors).

4.2.2 *Modelling and mapping to Jadex agents with Tropos4AS*

The **iCleaner** has been modelled with *Tropos4AS* using the TAOM4E tool, and the obtained models were mapped to a Jadex agent definition file (ADF) and Java code skeletons, by the *t2x* tool. The resulting ADF (>1200 lines of code, 16 goals and 36 plans, including auxiliary ones) contains the definition of the agent's belief base, available goals with their details, references to plans and accepted messages. The goal model structure, which guides the achievement of the high-level goals at run-time and alternatives selection, was implemented through the mapping process described above in this Section. The Java plans were manually implemented to realise the interface to the specific cleaner agent execution and testing environment which was developed for general agent testing by one of the case study participants.

The example in Fig. 11 shows a small part of the generated ADF for the *iCleaner* and the corresponding goal model part. The goal Observe_Environment is decomposed to two subgoals. This decomposition is annotated in the belief base (upper part of the figure) and handled by a dedicated plan. Plans (e.g. MoveToTarget) are handled by the Jadex goal triggering mechanism. Goal selection in means-end and OR-relationships is done by evaluation of softgoal contributions and the importance given to softgoals. For this system, we gave the same importance to each softgoal.

The interface to the simulation environment and classes representing the ontological concepts used in this environment were automatically generated, by tools provided with Jadex. Next, the concrete sensing and acting functionalities were implemented in the generated JAVA files.

4.2.3 *Test case generation and simulation*

We used the automated testing tool *eCat* [45] for testing the system for the achievement of its main goals, obtaining feedback to improve the design and requirements models.

For each version of the iCleaner on the simulation environment, 1000 test cases of 30 s each were performed, measuring the performance in terms of waste removed, obstacles hits and battery failures. The generated environments differ for the number and placement of waste, obstacles, charging stations and waste bins. The following measures were used over the 1000 test cases, to assess three adaptivity properties: Efficiency in collecting dirt, robustness as a measure for correctly avoiding obstacles, and failure, represented in how often the cleaner runs out of battery.

$$Robustness = Number\ of\ Crashes/Time \quad (1)$$

$$Efficiency = DirtCollected/Time \quad (2)$$

$$Failure = BatteryDrained/Time \quad (3)$$

4.2.4 *Requirements validation and refinement*

We iterated the process producing four versions of the **iCleaner**. Results of the preceding version were used as a feedback which gives rise to new requirements and bug reports that were taken into account in the development of the subsequent version. The **iCleaner** prototype exhibited basic adaptivity properties, which mainly arise from the interpretation of the extended goal model and the interplay of its goal types and conditions.

The behaviour of the system was improved, basing on quantitative (by measurements for efficiency and robustness) and qualitative (by observation) feedback from an automated testing of the implemented prototypes. This feedback led to changes in the goal models, in the detailed design and the implementation. Executing the requirements

model and guiding the agent behaviour directly with this goal model, wrong behaviours were localised, missing functionalities added and modifications enacted to the models and then mapped to the code.

In the first version of the *iCleaner* prototype generated with the *t2x* tool form the *Tropos4AS* model, upon a detailed analysis of the expected run-time behaviour for goal achievement and on the corresponding agent's plans, different lacks were identified, mainly regarding agent movement. The plans deciding the target location were scattered in different parts of the model. The decision on which target to select was made at a goal level, by defining inhibition links, to give precedence to battery loading. The resulting goal model leads to the following nominal run-time behaviour: the agent always tried to observe the environment, by locating new target destinations and moving to them. If there was some waste in the range of its cleaning tools, it was cleaned, either by absorbing or by mopping.

In the following revisions, upon feedback from the simulation, first the conditions were reworked to improve robustness, as can be seen in Fig. 12 which shows that a drastic reduction in crashes was monitored in *v*2. This came at the cost of a reduced efficiency. In version 3, the code of plans defining the agent movement towards battery loading stations and for emptying the cleaner's internal dust box into a waste bin were revised. Also, the code of plans for discovery of new charging stations and waste bins was improved by reusing a plan ExploreLeastSeenPlaces. This increased efficiency, but also failure due to an empty battery.

In version 4, to further improve efficiency, a new alternative MoveToNearestWaste to achieve the goal LocateNextTarget was added, which should be applied always if there exists waste that was sensed but not yet cleaned. Also, conditions were optimised to make the decision on next targets depend on the location of the

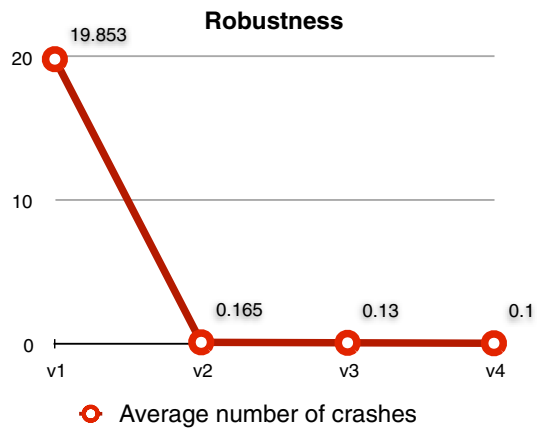

Fig. 12 Crashes (robustness)

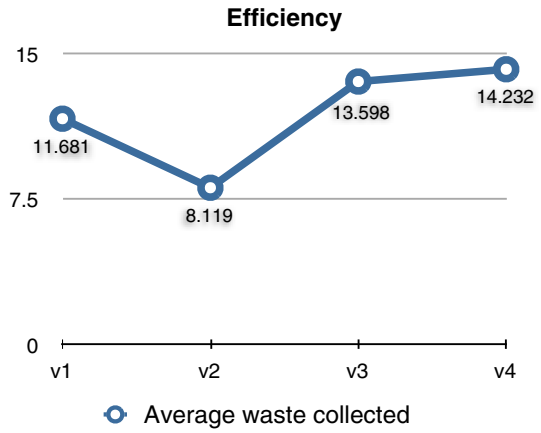

Fig. 13 Waste removed (efficiency)

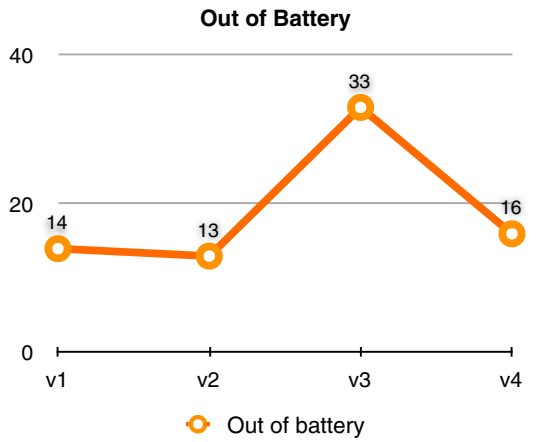

Fig. 14 Out of battery (failure)

loading station, when the battery is low battery, leading to the final results (*V*4) in Figs. 13 and 14.

The tool support helped maintain the consistency and traceability between goals and implemented functionalities along the development iterations of the prototype.

5 Empirical evaluation of Tropos4AS modelling

In the above section, we have shown the applicability of *Tropos4AS* at requirements-time when model validation and refinement tasks are performed. The results illustrate also how a self-adaptive behaviour at run-time can be driven by requirements encoded into an executable *Tropos4AS* model.

In this section, we describe an evaluation of the *Tropos4AS* modelling language in comparison with the underlying *Tropos* language[6] by conducting two controlled

[6] For this evaluation, we refer to the *Tropos* modelling language as defined in [9, 60] with the focus on *Tropos goal diagrams*, which are mainly affected by the proposed extensions.

Table 1 Overview of the two controlled experiments

	First experiment	Second experiment
Goal	Evaluate the effort and effectiveness in requirements modelling performed with the modelling languages *Tropos* and *Tropos4AS*	Evaluate the comprehensibility of requirements models expressed in the modelling languages *Tropos* and *Tropos4AS*
Objects	Textual requirements specifications of two adaptive systems: PMA, WMM	Requirements models of PMA/WMM in *Tropos* and *Tropos4AS*; and textual requirements specifications
Null-hypotheses	$H_0$1-The effort of modelling requirements with *Tropos4AS* is not significantly higher than the effort of modelling with *Tropos* $H_0$2-The effectiveness of *Tropos4AS* models is not significantly higher than the effectiveness of *Tropos* models	$H_0$3-The comprehensibility of system requirements cannot be significantly improved by using *Tropos4AS* models, in comparison with *Tropos* models
Main factor	Modelling language (treatments: *Tropos* and *Tropos4AS*)	Modelling language (treatments: *Tropos* and *Tropos4AS*)
Application domains	PMA and WMM	PMA and WMM
Experiment schedule	Overall ca. 4 h including preparation, Lab1 and Lab2	Overall 30′ including Lab1 and Lab2 (fixed laboratory times)
Dependent variables (measures)	Effort (*a*1-modelling effort; *a*2-perceived modelling effort; *a*3-modelling effort distribution; *a*4-difficulty in modelling), effectiveness (*a*5-perceived expressiveness; *a*6-perceived language expressiveness; *a*7-perceived language utility; *a*8-model accuracy) (*Labels correspond to column* ***Id*** *in Table 3*)	Model comprehension

experiments consisting of modelling and comprehension tasks performed by a group of twelve participants. The ultimate objective is to get empirical evidence about the trade-off between the modelling effort by the analyst who adopts *Tropos4AS* and the understanding of the system's adaptive properties brought by *Tropos4AS*.

The choice of using controlled experiments has been performed upon a careful analysis of the literature on empirical validation of visual modelling methods.

Indeed, despite the potential relevance, in the literature only few attempts exist to study and validate visual modelling methods through experiments. Various aspects complicate the definition of experimental measures and thus explain the limited application of empirical studies to RE methods, such as the involved iterative and time-consuming processes, and the informal and semi-formal languages used to reflect a *subjective* understanding of the domain [1]. For instance, [33] compared the two prominent goal modelling languages *i** and *KAOS* by an experiment in which they simulated a complete software development process, whereas [27] empirically evaluated the goal-oriented approach *Tropos* against scenario-based UML *Use Cases*. Different from these works, many RE methods are evaluated by performing case studies or by applying them to the modelling of a system, sometimes derived from industry [12, 24, 67]. This can be a strong limit for the use and adoption of new RE methods [10]. We present a design for a controlled empirical study that is feasible to be conducted and gives statistically accepted results and also a valuable insight to the practical use of the modelling languages, e.g. by experts and non-experts.

Experiments To understand whether a specific modelling language (here, *Tropos4AS*) brought improvements to the underlying general language (*Tropos*), we evaluate three main aspects: (1) the effort required for modelling; (2) the effectiveness of modelling (i.e. the expressiveness of models); and (3) the analyst's understanding of the system's adaptive properties that we investigate through a model comprehensibility task.

The goals of the two experiments are complementary: The **first experiment** aims at comparing the two modelling languages by asking the subjects to build and analyse some goal-oriented models, starting from a textual requirements description. Several measures of effort and effectiveness have been applied (row *Dependent variables (measures)* in Table 1). They include objective measures about time and "accuracy"[7] of the models built; and subjective measures concerning the perceptions of the participants in using the modelling languages. In the **second experiment**, we examined the comprehensibility of models created by applying the two languages.

The two experiments were executed following Wohlin's guidelines [68] and with 12 subjects: 6 research fellows and 6 PhD candidates working at FBK.[8] Part of the participants have industrial experience on using modelling

[7] Note that we do not aim to a formal representation of requirements for which correctness and accuracy can be verified, as e.g. in *Formal Tropos* [25].

[8] Fondazione Bruno Kessler (FBK) is a research centre located in Trento, Italy—see http://www.fbk.eu.

languages (such as UML and *Tropos*), while others have only basic knowledge.

The experiments are performed on two software systems that require adaptive behaviour and can be modelled with both the general and the specific languages: the PMA described above and a Washing Machine Manager (WMM). WMM is an intelligent washing machine controller, which adapts washing settings to user's preferences for energy saving and cleanness. Both systems provide normal as well as some exceptional behaviour, and some circumstances drive the system to change its behaviour, for satisfying its system requirements.

In both experiments, we adopted a *paired, counterbalanced* experimental design [68] based on two laboratory sessions and two different treatments (*Tropos* and *Tropos4AS*), and four groups of 3 people each, created by randomly dividing the 12 subjects. In this design, each subject performs the experimental tasks twice, each time working on a different software system and treatment. The design mitigates learning effects for treatments and objects and enables the application of more precise statistical methods, having the same number of subjects for each task assignment.

In the following, we give details about the design, execution and analysis of the results of the first experiment. The second experiment is instead detailed in [39] and only summarised in this section for a comprehensive discussion of the whole empirical evaluation.

5.1 First experiment

Focusing on modelling aspects in *Tropos4AS* and *Tropos*, we define the following two research questions.

> *RQ*1 How does the *effort* of modelling requirements with *Tropos4AS* compare to the effort of modelling them with *Tropos*?
> *RQ*2 Can the *effectiveness* of *Tropos4AS* models be compared to the effectiveness of *Tropos* models, for representing requirements of an adaptive system?

Each research question has been translated to the corresponding *null-hypothesis* H_0 and *alternative hypothesis* H_a. The *null-hypothesis* typically denotes that the treatment has no significant effect on the result. Only if statistical significance of the results is obtained, the null-hypothesis can be rejected in favour of the *alternative hypothesis*, which denotes that there exists an effect of the treatments on the result. We frankly expect that the *Tropos4AS* language requires a higher modelling effort than the original *Tropos* (e.g. it asks to explicitly model exceptional situations and conditions raising them), but we expect also that it improves the communication effectiveness of the models built, i.e. the models better reflect the requirements and increase the overall system comprehensibility. Therefore, we defined the research hypotheses with respect to *RQ*1 and *RQ*2 with the following directions (i.e. they are one-tailed):

- $H_0$1: The effort of modelling requirements with *Tropos4AS* is not significantly higher than the effort of modelling with *Tropos*.
- H_a1: The effort of modelling requirements with *Tropos4AS* is significantly higher than the effort of modelling with *Tropos*.
- $H_0$2: The effectiveness of *Tropos4AS* models is not significantly higher than the effectiveness of *Tropos* models.
- H_a2: The effectiveness of *Tropos4AS* models is significantly higher than the effectiveness of *Tropos* models.

5.1.1 Experiment procedure and variables

The experiment has been conducted individually by each participant in approx. 4 h, and it consisted of: (1) a 1-h tutorial about *Tropos* and *Tropos4AS* to be attend; (2) a pre-questionnaire to be filled, (3) a first session of laboratory concluded with a specific questionnaire, (4) a second session of laboratory concluded with a specific questionnaire and (5) a final overall post-questionnaire. Table 2 (third row) shows some of the questions we asked in Lab/Post-questionnaires.

The pre-questionnaire asks for the position (PhD student, researcher) of the participants and working experience, their knowledge and experience on the two methodologies. During the two laboratory sessions the participants had to model PMA and WMM, one with *Tropos*, and one with *Tropos4AS*, as assigned to them. We asked the participants to use paper and pencil, since modelling tools would introduce furthers threat to the experiment regarding training and usability. Each participant received a detailed description of the experiment procedure, the questionnaires and sequentially, for each of the two laboratories, a summary of the modelling language to use, the textual requirements specification of the system to model (WMM or PMA) and two control questions, useful to cross-check the models. The questionnaire associated with each laboratory includes questions that aim to capture the adequateness of the experiment objects and the time for the modelling task, and collects the subjective perceptions about the specific treatment applied, the actual time spent in each modelling activity (reading the specifications, looking up the guidelines, modelling). A post-questionnaire collects the users' feedback about the two languages.

Table 3 summarises the aspects investigated by means of the questionnaires, by research question, the type of the

Table 2 Fragments of objects and questions for the two experiments

First experiment: modelling
Fragment of PMA requirements specifications to model
System story summary: *The main aim of an "intelligent" patient monitoring agent (PMA) software is to ensure that a patient, in a "smart home" environment, follows the medical instructions on eating meals and taking medicine. The system should reduce the need for human assistance, but not bring annoyances to the patient's life. The system can access to reliable sensors that are able to measure if the patient takes the medicine and how often he eats. The flat is set up with loudspeakers in each room. It is also connected to the phone line for the system to request assistance from the care assistant*
Examples of questions in the Lab/Post-questionnaire
(Lab-Quest) *The effort of modelling seems too high for an efficient use in practice*
(Lab-Quest) *The obtained model is too abstract to be able to properly guide the programmers to an implementation respecting the requirements*
(Lab-Quest) *The concepts of the modelling language were detailed enough to model the requirements*
(Lab-Quest) *I had difficulties in modelling user preferences with contributions to softgoals*
(Post-Quest) *It was difficult to model the example, with all its details*
(Post-Quest) *I had enough time for accomplishing the modelling task*
Scenario 1 for the accuracy evaluation of PMA
"The patient did not eat breakfast. Now its time for lunch. Which plans will be executed? Suppose the patient will finally, also after a reminder, not eat lunch"
Expected result in terms of fundamental concepts: *monitor lunch eating*, *remind for lunch*, *"eat lunch" fails and "eat at least 2 meals" fails*, *call care assistant*
Second experiment: comprehension
Examples of comprehension and post-questionnaire questions for PMA
(Lab-Quest) *On which occasion can the dinner be skipped?*
(Lab-Quest) *With which sensors does the system have to interact to get the necessary information?*
(Lab-Quest) *The effort of modelling seems too high for an efficient use in practice*
(Post-Quest) *The comprehension questions were clear*
(Post-Quest) *I was able to extract the asked information from the goal model*

associated variables, value range and unit and the experimental task on which these variables were measured.

The variables associated with *RQ*1 aim to give an objective evaluation of the average effort required to model requirements specifications with *Tropos4AS* and *Tropos*. They measure the time required for modelling (aspect *a1* in Table 3), the participant's feeling about such an effort (*a2*, and how they perceived the effort and specific difficulties and limits due to the use of *Tropos4AS* (*a3* and *a4*).

For *RQ*2 on one hand we measure the participant's subjective feeling about the modelling language: (1) the expressiveness, in terms of concepts the language provides to describe adaptive systems (*a5* in Table 3); (2) the effectiveness, in terms of potential use of the models for implementation purposes (*a6*); and (3) the utility of each single basic construct of the language (*a7*). On the other hand, we get an expert evaluation of the accuracy (i.e. "correctness" with respect to previously defined scenarios) of the produced models (*a8*).

The investigated aspects represent the dependent variables of our experiment, while the modelling languages used (*Tropos4AS* or *Tropos*) represent the independent variable. Each aspect has been investigated with its own research question and associated hypotheses having the same directions than the ones of *RQ*1 and *RQ*2, following the template: H_0x: The aspect in modelling with *Tropos4AS* is not significantly better than the aspect in modelling with *Tropos*, and H_ax: The aspect in modelling with *Tropos4AS* is significantly better than the aspect in modelling with *Tropos*. All variables which denote an interval (i.e. a2, a4,...a7) have been measured by means of questionnaires based on a 5-point Likert scale (1: strongly agree, 2: agree, 3: (neutral) not certain; 4: disagree, 5: strongly disagree), used by participants to answer questions such as the ones shown in Table 2, third row. The time (for aspects a1 and a3) is self-measured by each participant in minutes spent on the tasks.

The evaluation of model accuracy (a8) was carried out by an expert not involved in the rest of the experiment, evaluating to what extent the model covers two key scenarios defined starting from the requirements specification of each of the experiment objects: the first scenario characterises some normal system behaviour, while the second one includes some exceptional occurrences. An example

Table 3 Main experiment variables and their association with the research questions *RQ1* and *RQ2*

Id	Aspect	Variable type	Values	Source
RQ1:Modelling effort				
a1	Modelling effort	Ratio (continuous)	{1...} sec.	Lab
a2	Perceived modelling effort			
a2.a	Overall effort	Interval	$\in$ {1,...,5}	Lab-Quest
a2.b	*Tropos4AS* additional effort	Interval	$\in$ {1,...,5}	Post-Quest
a3	Modelling effort distribution			
a3.a	Recalling modelling language specification	Ratio (continuous)	{1,...} sec.	Lab-Quest
a3.b	Understanding requirements specification	Ratio (continuous)	{1,...} sec.	Lab-Quest
a3.c	ACtual modelling	Ratio (continuous)	{1,...} sec.	Lab-Quest
a4	Difficulty in modelling			
a4.a	Difficulty of modelling	Interval	$\in$ {1,...,5}	Post-Quest, Lab-Quest
a4.b	Difficulty in using the modelling language	Interval	$\in$ {1,...,5}	Lab-Quest
RQ2 : Modelling effectiveness				
a5	Perceived language expressiveness			
a5.a	Effectiveness in capturing requirements	Interval	$\in$ {1,...,5}	Post-Quest
a5.b	Adequateness of the language concepts	Interval	$\in$ {1,...,5}	Post-Quest
a6	Perceived language effectiveness	Interval	$\in$ {1,...,5}	Lab-Quest
a7	Perceived language utility			
a7.a	Utility for modelling conditions	Interval	$\in$ {1,...,5}	Lab-Quest
a7.b	Utility for modelling failures	Interval	$\in$ {1,...,5}	Lab-Quest
a7.c	Overall language utility	Interval	$\in$ {1,...,5}	Lab-Quest
a8	Model accuracy	Interval	$\in$ {0,...,4}	Lab

Main variables are in bold

scenario together with the expected results for an evaluation can be found in the centre of Table 3. The expert analysed if the modelling concepts were correctly used and counts plans activated and goals failing in each scenario.

5.1.2 Data analysis and statistical evaluation

To analyse the results, one or more answers provided in the questionnaires were mapped to each aspect (a2 to a7). For each aspect, we then applied a statistical analysis to evaluate the corresponding null-hypothesis. The results were then grouped to answer *RQ*1 and *RQ*2. Some questions were posed for both treatments, and thus they have been directly compared with each other. The remaining questions have been compared with the "neutral" value on the Likert scale used.

Considering the nature of the variables (not necessarily normally distributed), the limited number of data points (two for each object), and the design of the experiment (paired and balanced, with both treatments applied to each subject), we selected the *nonparametric paired Wilcoxon test* [16] for evaluating each aspect for the two treatments (*Tropos4AS* and *Tropos*). A 5 % significance level was adopted for the obtained p value to determine whether the null-hypothesis can be rejected or not (in this case, no conclusion can be drawn neither on rejection nor on acceptance of the hypothesis). Hence, we rejected a null-hypothesis only if the probability that it is true, the p value, is smaller than 0.05 [68].

Furthermore, for each data set we computed average μ, median, and the *Cohen.d effect size* (for a paired design: $d = \frac{\mu_2 - \mu_1}{\sigma_D}$, where σ_D is the standard deviation of the pairwise differences) to analyse trends and to estimate the magnitude of the obtained result (as defined by Cohen [15], 0.2: small, 0.5: medium, 0.8: large effect).

Co-factors are all those factors different from the two treatments, which could possibly have an (undesired and/or uncontrolled) impact on the results of the experiment. We particularly tested if *object*, *subject experience* and *subject position* had a statistically significant impact on the results, by a two-way analysis of variance (ANOVA) test and a significance level of 0.05 for the obtained p value.

5.1.3 Results and interpretation

First, an evaluation of the adequateness of the experimental settings has been conducted by means of questions in the pre- and post-questionnaires asking whether the participants experienced any difficulty during the experiment,

whether they worked under time pressure, and whether object descriptions and tasks we asked to perform were clear. The participants' experience with the use of RE modelling languages reflects the presence of more experienced subjects (mostly researchers) and the less experienced ones (mostly PHD students). However, the initial understanding of both modelling languages was perceived as adequate and the participants did also not reveal particular difficulties in the use of both *Tropos4AS* and *Tropos*. Also, the tasks and objects of the experiment were considered to be adequate, well-understood and nearly equally difficult, for both languages. Therefore, we can claim that the experimental settings were adequate.

Table 4 shows the results obtained by comparing the values of the variables collected for both treatments (*paired* analysis) and the results for variables concerning only *Tropos4AS*, compared to the *neutral* response 3 in the 5-point Likert scale (*non-paired* analysis). In the following, we describe and c the single results, which were then aggregated to answer the research questions.

5.1.4 Findings about RQ1: effort

(a1) Fig. 15 (centre) contains a boxplot for the overall time spent by the participants for the modelling task (a1). It shows that modelling requirements specifications with *Tropos4AS* **requires more time** than modelling them with *Tropos* (75 vs. 49.5 min for the median). With a p value of 0.018, the result is statistically significant, and thus the null-hypothesis H_0a1 can be rejected. Intuitively, this can be explained by the richer models obtained with *Tropos4AS*.

(a2) However, the **participants perceive that the effort of modelling with *Tropos4AS* is not particularly higher** than with *Tropos* (a2.a). This fact cannot be proven statistically (we cannot reject H_0a2), but the medians (3 for *Tropos* vs. 3.5 for *Tropos4AS*) show a quite similar trend. We can speculate that giving the possibility for e.g. an explicit modelling of conditions and of the exceptional flows will give the possibility to express the requirements in a more intuitive way and thus decreases the perceived modelling effort. Moreover, the participants agree that it is worth to put additional effort in modelling details of the requirements with *Tropos4AS*, with a median of 2 (a2.b) and statistical significance.

(a3) Tropos4AS requires **more effort for reading the language specification (a3.a), while no statistically significant difference exists for the other activities (a3.b, a3.c)**, so we cannot reject H_0a3. The result is also confirmed by medians and Cohen-d effect sizes: for reading and understanding the example and for modelling it, the time difference is negligible (11 % in reading: median of 24.6 min for *Tropos* vs. 27.3 for *Tropos4AS*; 3 % in modelling: 21.3 min for *Tropos* vs. 23.25 for *Tropos4AS*). The time averages show the same trend: a huge difference in reading the language specification, but limited differences for requirements reading and modelling.

(a4) Tropos4AS **seems not to give more difficulties** than *Tropos* for modelling the requirements of the object. Also this result cannot be confirmed statistically, we cannot reject H_0a4, but by the average the participants perceived the same or even slightly less difficulty in using *Tropos4AS* than *Tropos*. This result can be explained taking into account further comments of participants: the additional

Table 4 Statistical analysis: comparison Tropos versus Tropos4AS

Aspect	Paired analysis?	Median *Tropos*	Median *Tropos4AS*	Reject null-HP?	p value	Cohen-d effect size
a1 [min]	Y	49.5	75	Y	0.018	0.83 (large)
a2.a	Y	3.5	3	N	0.6	0.18 (negligible)
a2.b	N	–	2	Y	0.0043	1.16 (large)
a3.a [min]	Y	3.65	9.35	Y	0.0046	0.87 (large)
a3.b [min]	Y	24.6	27.3	N	0.4	0.21 (small)
a3.c [min]	Y	21.3	23.25	N	0.47	0.29 (small)
a4.a	Y	3	3	N	0.88	0 (small)
a4.b	Y	3	3	N	0.42	0.27 (small)
a5.a	Y	4	2	Y	0.023	1.7 (large)
a5.b	Y	3	2.5	N	0.2	0.42 (small)
a6	Y	4	2	Y	0.0039	0.68 (medium)
a7.a	N	–	2	Y	0.002	1.6 (large)
a7.b	N	–	2	Y	0.006	1.04 (large)
a7.c	N	–	2	Y	0.001	2.1 (large)
a8	Y	3	4	Y	0.005	0.7 (medium)

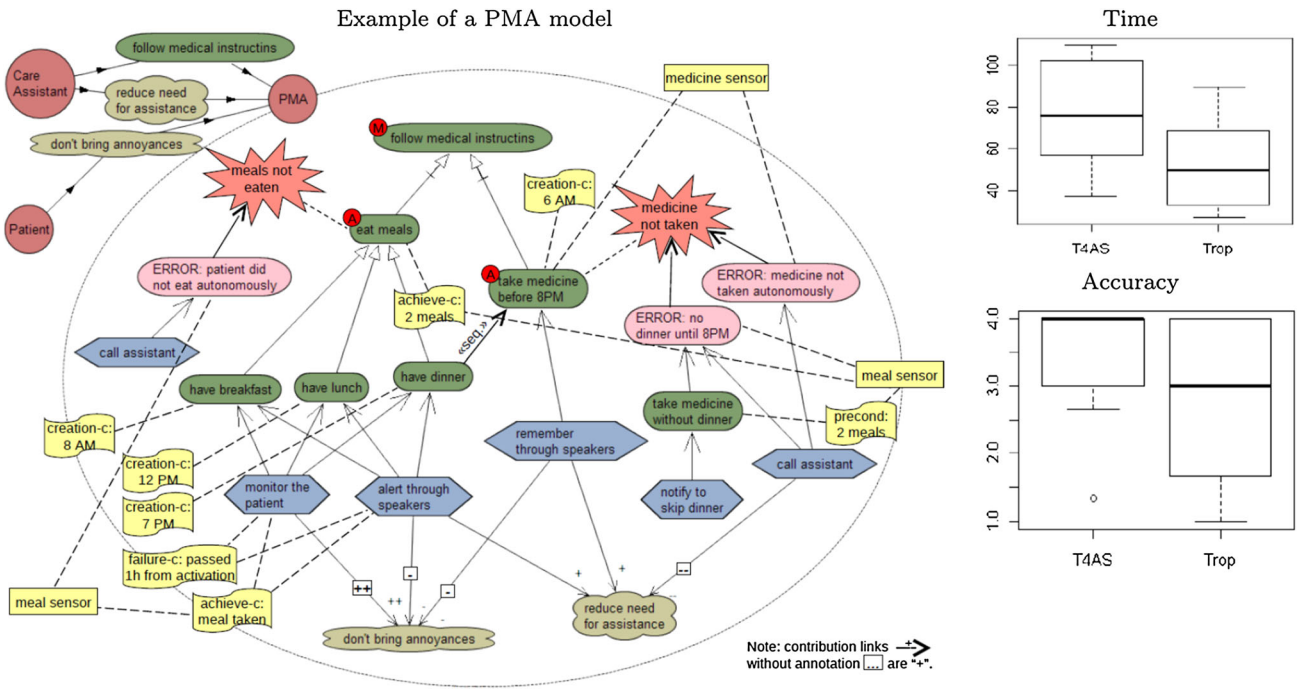

Fig. 15 Modelling task: a sample model to give the reader an idea about the models built by participants, and boxplots for modelling time (in minutes) and model accuracy (max. 4 points per scenario)

modelling concepts introduced with *Tropos4AS* seem not only to bring higher complexity, but also to facilitate expressing the modelling intentions.

Overall, we have to answer in affirmative way to the research question *RQ*1, hence we conclude: *Yes, the effort required to apply Tropos4AS is higher than the effort required to apply Tropos. However, the additional effort is not perceived by the users as such. They do not face particular difficulties and spent significantly more time only for studying the new Tropos4AS modelling concepts.*

5.1.5 Findings about RQ2: effectiveness

(a5) The participants agree that *Tropos4AS* produces models that are **more expressive** than *Tropos* models (a5.a). Moreover, the medians and the effect size show a trend that the participants were more confident for *Tropos4AS* than for *Tropos* and that the concepts of the modelling language are detailed enough for modelling the requirements. However, the relative null-hypothesis H_0a5 cannot be rejected, (a5.b).

(a6) Tropos4AS is perceived to be **more effective** than *Tropos* for producing models that are concrete and can guide the developers to the implementation. H_0a6 can be rejected with statistical significance.

(a7) The subjects agree that enriching *Tropos* is **useful** for modelling adaptive systems (in general and also in particular for conditions and failures). H_0a7 can be rejected.

(a8) The scenario-based analysis of model accuracy conducted by the expert shows with statistical evidence that the models produced with *Tropos4AS* are **more accurate** than the models produced with *Tropos*. H_0a8 can be rejected (a8) (an example of *Tropos4AS* model obtained from one participant for PMA, transcribed from the original and slightly cleaned it for a better representation, is shown in Fig. 15 left). *Tropos4AS* models covered the evaluation scenarios by more than 87 %, while the *Tropos* models covered the scenarios only by 66 % of relevant concepts selected by the expert for the considered scenarios and objects (Fig. 15 right).

Overall, we can thus answer in an affirmative way to the research question *RQ*2 and conclude that: *Yes, Tropos4AS allows the users to produce models more effective than Tropos for representing requirements of an adaptive system.*

5.1.6 Additional results

An additional analysis of the results shows various findings that are not directly related to the research questions, but important for understanding how *Tropos* and *Tropos4AS* are used in practice, by both experienced and novice software engineers. The participants spent from 27 to 90 min for the *Tropos* assignment, and from 37 to 110 min for *Tropos4AS*, with high, but very similar variances. On average, for both treatments together, the participants spent

47 % of their time with the reading of requirements and only 42 % with modelling. 11 % were used for looking at the language specification. This time could be reduced with better training. Specifically for *Tropos4AS*, on average 59 % of the modelling time was used for goal modelling, 25 % for conditions modelling and only 16 % for failure modelling.

To get an additional indicator of model complexity, we gave a look on the size of the models drawn during the experiments. They had a median of 19 entities and 28 relationships for *Tropos* models, and a median of 30 entities and 35 relationships for *Tropos4AS* models. Out of them, 17 entities and 19 relationships are *Tropos* concepts. These numbers can give an idea that the use of the additional *Tropos4AS* concepts increased the model size, but also reduced the need for using *Tropos* relationships to model complex behaviours, as it was also pointed out in some of the participants' comments.

Furthermore, the participants used most of the extensions according to the modelling philosophy, but had some difficulties with the semantics of the different types of condition and goals, truly due to the short training. However, as we have seen in (a8) the accuracy of the models created with *Tropos4AS* is significantly higher, compared to *Tropos* models.

To exclude an undesired impact on the experiment validity, three co-factors were analysed for the presence of a statistically significant interaction with the experiments' treatments: the subjects' working *position*, the subject *experience* in using RE methods and modelling languages, and the *objects* (WMM, PMA). An ANOVA test could reveal only a very limited impact of co-factors to the experiment. The only statistically relevant result refers to participants having experience in using RE methods (in general, despite the treatment). They produce more correct models than other participants (p value 0.04). Furthermore, this group of participants perceives less effort in modelling (p value 0.0049) and gives stronger agreement to the claim that the concepts of the *Tropos4AS* modelling languages are detailed enough to model the requirements (p values 0.02 and 0.024). However, no statistically significant impact can be seen considering the *interaction between experience and treatment*. This can be explained by the random, even distribution of the experienced participants in the experiment groups.

5.2 Summary of results for the second experiment

To provide a comprehensive overview on the study here, we recall the main results of the second experiment, whose aim is to evaluate the comprehensibility of models created by applying the two modelling languages [39], giving answer to:

*RQ*3 How does the comprehension of models created with *Tropos4AS* compare to the comprehension of models created with *Tropos*?

The experiment was performed three months after the first one, with the same participants. The overall duration was about thirty minutes, during which several comprehension questions were answered in a concise, fixed time using both a model (either *Tropos* or *Tropos4AS*) and the corresponding textual requirements specification. Examples are given in Table 2. Participants were also asked to fill a post-questionnaire with additional questions to strengthen the results of the study, and questions devoted to assess the validity of the experiment set-up.

The accuracy of the answers provided by participants to the comprehension questions was assessed by computing *precision*, *recall* and *f-measure* for each participant's answer with respect to the gold-standard answers provided by an expert. The statistical analysis performed on the collected data exploits the same paired *Wilcoxon* test as in the modelling experiment.

Figure 16 shows boxplots of the obtained results for precision, recall and f-measure. We observe that *Tropos4AS* significantly improved both precision and recall, and consequently the f-measure, of the participant answers (referring to the median, respectively, form 70 to 90 %, form 58 to 75 % and form 65 to 81 %). These results are statistically relevant (p value $<5\,\%$) and supported by medium effect size. Therefore, we can claim with statistical evidence that a *Tropos4AS* model, together with the textual specifications, is more effective for retrieving accurate information than a *Tropos* model, together with the same textual specifications.

To check whether the information to answer the questions was extracted from the models or from the textual specifications, we analyse the post-questionnaire, which gives a self-assessment about the use of the model and textual specifications. The analysis shows with statistical significance that the amount of information extracted from *Tropos4AS* models is quantitatively larger than from

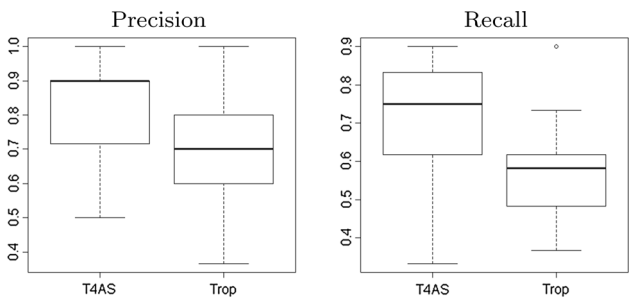

Fig. 16 Boxplots for the distribution of the averages per participant, for precision and recall of the single answers to the comprehension questions

Tropos models. Similarly, the amount of information extracted from the textual requirements is quantitatively lower when using *Tropos4AS* than when using *Tropos* models. A total of 9 out of 12 subjects (75 %) stated that using *Tropos* they needed to extract a substantial amount of information from the textual specifications to answers to the comprehension questions, compared to 2 out of 12 for *Tropos4AS*.

These results confirm that *Tropos4AS* is more effective than *Tropos* to support retrieving accurate information from the requirements specifications of adaptive systems. We hence give positive answer to *RQ*3: *The comprehensibility of system requirements can be significantly increased by using Tropos4AS models, in comparison with Tropos models*.

An analysis of various co-factors did not reveal any statistically relevant impact of participant experience, participant working position, and objects in the achieved results. We could, however, observe that participants with high experience in *Tropos* did not significantly increase their performance by the use of *Tropos4AS*. Instead, less experienced subjects' performance was more heavily influenced by the use of *Tropos4AS*, improving by 20–25 % the correctness of their comprehension task.

5.3 Overall findings of the Empirical Study

Observing the results achieved, we can conclude that the effort required to apply *Tropos4AS* is higher than the effort required to apply *Tropos* (RQ1). However, the additional time required by *Tropos4AS* was spent mainly for learning the language syntax and semantics. In fact, this additional effort is not perceived as such by the participants, and they did not face particular difficulties in using *Tropos4AS*. *Tropos4AS* allows the participants to produce models more effective and accurate than *Tropos* for representing requirements of an adaptive system (RQ2), and the models obtained contribute better to the comprehension of system requirements (RQ3), especially for novice users.

The results of this empirical evaluation met our expectations and strengthen them. In particular, the use of *Tropos* and the *Tropos4AS* extensions by novice users was better than expected and their comments show that the additional constructs bring few more complexity, but facilitate expressing a set of requirements of simple adaptive systems such as the examples used. This was confirmed, from a different viewpoint, by the comprehension experiment.

5.4 Threats to validity

The study was performed with a balanced design with both treatments applied to each participant, to limit the learning effect and the impact of the participants' experience. This design allows to use proper statistical tests to validate the null-hypotheses. Furthermore, to limit the learning effect on the study we organised a pre-experiment tutorial in which we introduced *Tropos* and *Tropos4AS* to the participants. To limit the authors' bias, the design and conduction of the experiments were carried out by the author not involved in the definition of the *Tropos4AS* modelling language. The objects under study (i.e. the systems to model) are fragments of real adaptive systems, even if not trivial—in fact, they required a quite high understanding and modelling effort. The use of more complex objects would not have been treatable in an empirical study of such extent. An evaluation of the framework in an industrial case study, however, would give complementary results to those obtained.

To limit potential threats caused by a low number of available subjects, in our experiment we: (1) involved only experienced subjects, avoiding to include junior students; (2) adopted a paired, counterbalanced experimental design [68]: that is we involved 12 subjects and we asked each of them to work with the two considered treatments (i.e. modelling languages), and thus we overall collected 24 data points; and (3) used nonparametric statistical tests to analyse the collected data: that is we used tests known to adequately work also with small numbers [61, 65]. In fact, these tests can be used when the data is not of sufficient quality to satisfy the assumptions of parametric tests, which are preferred when the assumptions are met because they are more sensitive.

The evaluation of the variables we collected was both objective and subjective. For the variables measured on a 5-point Likert scale we exclusively relied on the participants' perceptions and opinions while with the evaluation of models and the answers provided to the comprehension questions we obtained more objective measures. In line with studies in the literature (e.g. [55]) we asked an expert having a 10-year experience in the use of RE methods and requirements specification modelling to define a gold standard (in terms of both "optimal" models and "correct" answers) and to contrast the models drawn by each participant with it, by objective measures. We consider an expert-based evaluation to be in between a subjective evaluation (i.e. subjects give their opinion about modelled system properties) and an objective evaluation (e.g. objective metrics are used to measure system design properties). An objective evaluation of the accuracy of models produced by humans for capturing system requirements (de-facto strongly human-oriented artefacts) cannot be easily conducted since different and complex (human and subjective) factors influence both the model properties [1, 36, 58] and their evaluation. In such a case, an appealing option consists in involving one (or more) recognised experts. The expert is asked to produce the

artefact according to his experience and then this artefact is compared (following specific and predefined objective criteria) with the ones produced by the experimental subjects.

To encourage a repetition of the studies with different participants, the experiment packages for both experiments are available online[9]

6 Related work

Several approaches have been proposed in the last years to address the challenges in requirements engineering for self-adaptive systems that were pointed out in the already mentioned research agendas[12, 20, 22, 56, 57]. Most of them are analysed in a systematic literature review [69]. In this section, we discuss those that, analogously to *Tropos4AS*, attempted at addressing issues related to uncertainty at requirements-time, synchronisation between requirements and architecture, and requirements as run-time objects.

The requirements language RELAX [67] provides the analyst with the possibility to specify "relaxable" requirements through the use of a vocabulary ("may", "close to", etc.) that expresses need of flexibility, either in the degree of satisfaction of a requirement or in the ways a requirement can be implemented. Goal-oriented analysis, such as the *KAOS* obstacle analysis ([63]), is used [13] to identify alternative options for the achievement of a requirement. Requirements elicitation as supported by the RELAX approach could be used in *Tropos4AS*, which in addition covers later phases of software engineering.

A similar approach [4] exploits *KAOS* to represents requirements for adaptive systems that can admit small or transient fulfilment violations, which do not compromise the intended behaviour of a system, e.g. restarting a washing machine that stopped unexpectedly, without re-executing the washing program since the beginning. The concepts of *fuzzy* goals and constraints are introduced to capture a certain degree of uncertainty, while the concept of *adaptive* goals allows defining run-time countermeasures in a parametric way, with respect to the satisfaction level of other goals. These approaches address uncertainty at requirements-time by relaxing goal satisfaction criteria using fuzzy logic, while in *Tropos4AS* uncertainty management is performed through modelling alternative behaviours, associated with environment conditions and qualities. The *Tropos4AS* goal model component supports decisions of the system at run-time, which is driven by the optimisation of softgoals together with the continuous evaluation of conditions of the environment.

Various works share with *Tropos4AS* the idea of using goal-oriented modelling for variability analysis at requirements-time. [26] and [7] use, respectively, *i** and *KAOS*, and propose to model each possible system configuration in a distinct goal model, together with the conditions for transition between these configurations. In contrast, our approach captures the variability in a single model, which simplifies model evolution and consistency, especially during the earlier development phases. In [52] which proposes a modelling language based on *Techne* [29], the notion of *adaptive* requirements is used to express functional and non-functional properties whose specification rests on a set of variants, each one associated with specific activation conditions, and which can lead to different satisfaction degrees for qualities and preferences. Complex goal achievement behaviours can, however, not be captured.

Similarly, [59] proposes the notion of *awareness requirement*, to be used for complementing a given requirements model with a specification of feedback loop mechanisms and indicators of success rate in requirements fulfilment. This specification is implemented by event-condition-action rules in an ad hoc execution framework, while we propose to use a BDI-agent-based implementation of the behaviour for requirements simulation, but do not tie to a specific implementation framework.

As in *Tropos4AS*, other works extend goal-oriented models to enrich their expressiveness, thus enabling to express the environment or contextual parameters and their monitoring, when modelling adaptive systems. [32] enriches *i** models with various annotations and variable contribution to softgoals. The enriched language can model the environment and its influence on the system behaviour only at a high-level, through *i** delegation. Unlike *Tropos4AS*, with this approach the run-time goal satisfaction behaviour cannot be specified.

The goal decomposition tree (GDT) approach [34] allows specifying and validating the run-time behaviour of an agent situated in a dynamic environment, by *lazy* and necessary satisfaction conditions and guaranteed properties in case of failure. GTD includes a translation to executable code, but is conceived for a formal specification of software and does not define a methodology. Similarly, [21] defines a goal model for dynamic systems (GMoDS), including goal precedence and triggering relationships implemented by assigning the leaf goals to agent roles in the MAS. By this process, high-level information on goal decomposition and alternatives is difficult to be traced back at run-time. Moreover, the provided MAS simulation environment does not make use of goal-oriented technology.

[9] The questionnaires, a detailed description and the anonymised raw data of the experiments are available at http://selab.fbk.eu/morandini/taom4e_eval/.

The problem of synchronising system requirements and architecture is addressed in [53] that proposes an architecture to enable requirements-driven adaptation of service-based applications. It exploits event-condition-action rules to represent requirements artefacts as run-time objects. Differently, in *Tropos4AS* the run-time representation of requirements is given using executable goal models, thus preserving the goal-oriented specification at run-time.

Ideas similar to ours are considered in [17], putting the focus on the reconfiguration of distributed agent systems and the presentation of a run-time architecture, while *Tropos4AS* has its main focus on requirements engineering, and the adaptation of single agents.

In [18] run-time goal models are introduced, which are specifically annotated with achievement rules, while in *Tropos4AS* we give semantics to the different goal types, which are intended for requirements analysis and run-time phases. The proposal in [14] combines requirements-driven adaptation by reasoning on goal models, and architecture-driven adaptation. They propose a framework for run-time adaptation, which considers the requirements goal model and a model of design decisions, to create a new run-time model, thus being in part complementary to our work, which focuses on modelling adaptivity and verification of modelled behaviours at a requirements level.

Run-time adaptation by providing to the system (or to the platform it is deployed on) mechanisms to reason on requirements at run-time is also investigated in recent research, which borrow knowledge representation and reasoning techniques developed in the Artificial Intelligence field. For instance, Bayesian network techniques are used in [5], and case-based reasoning techniques in [51]. These types of run-time reasoning on requirements could be integrated with those provided by *Tropos4AS* to overcome the limits of run-time adaptation based on the selection among pre-defined behaviours and pave the way towards addressing online evolution of software application.

7 Conclusion

In this paper, we gave a full account of the *Tropos4AS* framework for engineering requirements of adaptive systems. *Tropos4AS* combines in a single framework abstractions and techniques from different paradigms, namely goal-oriented requirements engineering, agent-oriented software engineering and BDI-agent software platforms. Moreover, it paves the way to a complete formal binding between the intended goal model semantics at requirements engineering time and the run-time behaviour. Developing the *Tropos4AS* framework, we addressed main challenges highlighted in various research agendas for adaptive systems, [6, 20, 22], specifically those concerning requirements-time uncertainty, the need of a run-time representation of requirements and the synchronisation of requirements with the system architecture. A tool-supported mapping of *Tropos4AS* requirements models to *Jadex* BDI agents has been introduced, but the requirements models can be straightforwardly used for other agent and object-oriented languages, by implementing a middleware layer with monitoring and goal evaluation functionalities.

The applicability of prototype simulation of *Tropos4AS* models for requirements validation and refinement is illustrated through the iCleaner case study.

Furthermore, the paper presented an empirical evaluation of the *Tropos4AS* modelling language, consisting of two controlled experiments, aimed at investigating the comprehensibility, effectiveness and modelling effort in comparison with *Tropos*. We described its experimental design and execution and discussed the results, which show that *Tropos4AS* is more effective than *Tropos* in describing requirements of adaptive systems, especially when used by novices. The experiment design can be reused for evaluating the performance of other conceptual modelling languages that have been defined as extensions for a specific domain. The positive results indicate that more specific empirical studies can be conduced to evaluate particular aspects and properties of *Tropos4AS* or to characterise the adoption of *Tropos4AS* in specific contexts and for specific purposes (e.g. system implementation and detection of design flaws).

We believe that *Tropos4AS* provides opportunities for further investigation on software evolution motivated by requirements changes. Supporting the systems' users to express requirements changes with respect to the existing requirements models, and to motivate the extension with new system capabilities, will be a possible follow-up of this work. Another aspect to be considered in future is the adaptation and optimisation of the software behaviour by cooperation of multiple agents in the system.

Additionally, a formalisation of proactive behaviours and the exploitation of machine learning techniques to enable proactivity and a dynamic modifications of goal- and failure-models on the basis of the execution history seems worth to be investigated for run-time evolution.

References

1. Aranda J, Ernst N, Horkoff J, Easterbrook S (2007) A framework for empirical evaluation of model comprehensibility. In: ICSE workshop on modeling in software engineering (MISE '07), p 7

2. Asnar Y, Bryl V, Giorgini P (2006) Using risk analysis to evaluate design alternatives. In: AOSE, pp 140–155
3. Avizienis A, Laprie JC, Randell B, Landwehr CE (2004) Basic concepts and taxonomy of dependable and secure computing. IEEE Trans Dependable Sec Comput 1(1):11–33
4. Baresi L, Pasquale L, Spoletini P (2010) Fuzzy goals for requirements-driven adaptation. In: Proceedings of the 2010 18th IEEE international requirements engineering conference, IEEE Computer Society, pp 125–134
5. Bencomo N, Belaggoun A, Issarny V (2013) Dynamic decision networks for decision-making in self-adaptive systems: a case study. In: 8th International symposium on software engineering for adaptive and self-managing systems SEAMS, San Francisco, pp 113–122
6. Bencomo N, Whittle J, Sawyer P, Finkelstein A, Letier E (2010) Requirements reflection: requirements as runtime entities. In: ICSE 2010, vol 2, pp 199–202
7. Berry D, Cheng B, Zhang J (2005) The four levels of requirements engineering for and in dynamic adaptive systems. In: Proceedings of the 11th international REFSQ workshop, Porto, Portugal
8. Braubach L, Pokahr A, Moldt D, Lamersdorf W (2004) Goal representation for bdi agent systems. In: PROMAS, pp 44–65
9. Bresciani P, Giorgini P, Giunchiglia F, Mylopoulos J, Perini A (2004) Tropos: an agent-oriented software development methodology. Auton Agents Multi-Agent Syst 8(3):203–236
10. Brun Y (2010) Improving impact of self-adaptation and self-management research through evaluation methodology. In: Proceedings of software engineering for adaptive and self-managing systems (SEAMS10), pp 1–9
11. Cailliau A, van Lamsweerde A (2012) A probabilistic framework for goal-oriented risk analysis. In: Proceedings of the 20th IEEE international conference on requirements engineering (RE 2012). IEEE
12. Cheng BHC, de Lemos R, Giese H, Inverardi P, Magee J (eds) (2009a) Software engineering for self-adaptive systems (outcome of a Dagstuhl Seminar). Lecture notes in computer science, vol 5525. Springer
13. Cheng BHC, Sawyer P, Bencomo N, Whittle J (2009b) A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In: MODELS '09: 12th International conference on model driven engineering languages and systems. Springer, pp 468–483
14. Chen B, Peng X, Yu Y, Nuseibeh B, Zhao W (2014) Self-adaptation through incremental generative model transformations at runtime. In: 36th international conference on software engineering, ICSE '14. Hyderabad, pp 676–687
15. Cohen J (2004) Statistical power analysis for the behavioral sciences. Lawrence Erlbaum Associates, Hillsdale
16. Dalgaard P (2008) Introductory statistics with R (statistics and computing), 2nd edn. Springer, Berlin
17. Dalpiaz F, Giorgini P, Mylopoulos J (2013) Adaptive socio-technical systems: a requirements-based approach. Requir Eng 18(1):1–24
18. Dalpiaz F, Borgida A, Horkoff J, Mylopoulos J (2013) Runtime goal models: Keynote. In: IEEE 7th international conference on research challenges in information science, RCIS 2013. Paris, pp 1–11
19. Dastani M, van Riemsdijk MB, Meyer JJC (2006) Goal types in agent programming. In: ECAI, pp 220–224
20. de Lemos R, Giese H, Müller HA, Shaw M et al (2010) Software engineering for self-adaptive systems: a second research roadmap. In: de Lemos R, Giese H, Müller HA, and Shaw M (eds) Software engineering for self-adaptive systems II. Lecture Notes in Computer Science, vol 7475. Springer, Berlin, pp 1–32
21. DeLoach SA, Miller M (2009) A goal model for adaptive complex systems. In: International Conference on Knowledge-Intensive Multi-Agent Systems (KIMAS 2009), St. Louis, MO
22. Di Nitto E, Ghezzi C, Metzger A, Papazoglou MP, Pohl K (2008) A journey to highly dynamic, self-adaptive service-based applications. Autom Softw Eng 15(3–4):313–341
23. Duff S, Harland J, Thangarajah J (2006) On proactivity and maintenance goals. In: AAMAS '06: proceedings of the fifth international joint conference on autonomous agents and multi-agent systems. ACM, New York, pp 1033–1040
24. Estrada H, Rebollar AM, Pastor O, Mylopoulos J (2006) An empirical evaluation of the *i** framework in a model-based software generation environment. In: CAiSE, pp 513–527
25. Fuxman A, Liu L, Mylopoulos J, Roveri M, Traverso P (2004) Specifying and analyzing early requirements in tropos. Requir Eng 9(2):132–150
26. Goldsby HJ, Sawyer P, Bencomo N, Hughes D, Cheng BHC (2008) Goal-based modeling of dynamically adaptive system requirements. In: ECBS 08, Belfast, Northern Ireland
27. Hadar I, Kuflik T, Perini A, Reinhartz-Berger I, Ricca F, Susi A (2010) An empirical study of requirements model understanding: use case vs. Tropos models. In: SAC, pp 2324–2329
28. Hindriks KV, van Riemsdijk MB (2007) Satisfying maintenance goals. In: DALT, pp 86–103
29. Jureta IJ, Borgida A, Ernst NA, Mylopoulos J (2010) Techne: towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling. In: 18th IEEE International requirements engineering conference. Sydney, pp 115–124
30. Kephart JO, Chess DM (2003) The vision of autonomic computing. IEEE Computer 36(1):41–50
31. Kotsiantis SB (2007) Supervised machine learning: a review of classification techniques. In: Conference on emerging artificial intelligence applications in computer engineering. IOS Press, Amsterdam, pp 3–24
32. Lapouchnian A, Yu Y, Liaskos S, Mylopoulos J (2006) Requirements-driven design of autonomic application software. In: CASCON, pp 80–94
33. Matulevicius R, Heymans P (2007) Comparing goal modelling languages: an experiment. In: REFSQ, pp 18–32
34. Mermet B, Simon G (2009) GDT4MAS: an extension of the GDT model to specify and to verify multiagent systems. In: AAMAS'09, pp 505–512
35. Molesini A, Omicini A, Viroli M (2009) Environment in agent-oriented software engineering methodologies. Multiagent Grid Syst 5(1):37–57
36. Moody D (2007) What makes a good diagram? improving the cognitive effectiveness of diagrams in is development. Adv Inf Syst Dev pp 481–492
37. Morandini M (2011) Goal-oriented development of self-adaptive systems. PhD thesis, DISI, Università di Trento, Italy. http://eprints-phd.biblio.unitn.it/511
38. Morandini M, Dalpiaz F, Nguyen C, Siena A (2014) The tropos software engineering methodology. In: Cossentino M, Hilaire V, Molesini A, Seidita V (eds) Handbook on agent-oriented design processes. Springer, Berlin, pp 463–490
39. Morandini M, Marchetto A, Perini A (2011) Requirements comprehension: a controlled experiment on conceptual modeling methods. In: Proceedings of the first workshop on empirical requirements engineering (EmpiRE11)
40. Morandini M, Migeon F, Gleizes MP, Maurel C, Penserini L, Perini A (2009a) A goal-oriented approach for modelling self-organising MAS. In: Proceedings of the 10th international workshop on Engineering Societies in the Agents' World (ESAW 2009). LNCS, vol 5881. Springer
41. Morandini M, Penserini L, Perini A (2008) Automated mapping from goal models to self-adaptive systems. In: ASE 2008 Demo session, pp 485–486

42. Morandini M, Penserini L, Perini A (2008) Modelling self-adaptivity: a goal-oriented approach. In: 2nd IEEE international conference on self-adaptive and self-organizing systems (SASO'08). IEEE, pp 469–470
43. Morandini M, Penserini L, Perini A (2008) Towards goal-oriented development of self-adaptive systems. In: SEAMS '08: workshop on software engineering for adaptive and self-managing systems. ACM, New York, pp 9–16
44. Morandini M, Penserini L, Perini A (2009) Operational semantics of goal models in adaptive agents. In: 8th International conference on autonomous agents and multi-agent systems (AAMAS), IFAAMAS
45. Nguyen CD, Perini A, Tonella P (2008) eCAT: a tool for automating test cases generation and execution in testing multi-agent systems (demo paper). In: 7th International conference on autonomous agents and multiagent systems (AAMAS), IFAAMS, pp 1669–1670
46. Nguyen CD, Perini A, Tonella P (2010) Goal-oriented testing for mass. Int J Agent Oriented Softw Eng 4(1):79–109. doi:10.1504/IJAOSE.2010.029810
47. Omicini A, Ricci A, Viroli M (2006) Agens Faber: toward a theory of artefacts for MAS. Electron Notes Theor Comput Sci 150(3):21–36
48. Padgham L, Winikoff M, DeLoach SA, Cossentino M (2008) A unified graphical notation for aose. In: AOSE, pp 116–130
49. Penserini L, Perini A, Susi A, Mylopoulos J (2007) High variability design for software agents: extending tropos. ACM Trans Auton Adapt Syst (TAAS) 2(4):16–25
50. Perini A, Susi A (2004) Developing tools for agent-oriented visual modeling. In: Lindemann G, Denzinger J, Timm I, Unland R (eds) Multiagent system technologies. Proceedings of the second German conference, MATES 2004. LNAI, vol 3187. Springer, pp 169–182
51. Qian W, Peng X, Chen B, Mylopoulos J, Wang H, Zhao W (2014) Rationalism with a dose of empiricism: case-based reasoning for requirements-driven self-adaptation. In: Proceedings of RE'14. Karlskrona (SE). IEEE, pp 113–122
52. Qureshi NA, Jureta I, Perini A (2012) Towards a requirements modeling language for self-adaptive systems. In: Regnell B, Damian DE (eds) REFSQ. LNCS, vol 7195. Springer, pp 263–279
53. Qureshi N, Perini A (2010) Continuous adaptive requirements engineering: an architecture for self-adaptive service-based applications. In: RE@RunTime workshop at RE'10. Australia, Sydney, pp 17–24
54. Rao AS, Georgeff MP (1995) Bdi agents: from theory to practice. In: ICMAS, pp 312–319
55. Ricca F, di Penta M, Torchiano M, Tonella P, Ceccato M (2010) How developers' experience and ability influence web application comprehension tasks supported by uml stereotypes: a series of four experiments. IEEE Trans Softw Eng 36(1):96–118
56. Salehie M, Tahvildari L (2009) Self-adaptive software: landscape and research challenges. ACM Trans Auton Adapt Syst (TAAS) 4(2):14–42
57. Sawyer P, Bencomo N, Whittle J, Letier E, Finkelstein A (2010) Requirements-aware systems: a research agenda for RE for self-adaptive systems. In: IEEE International conference on requir Engeneering (RE10), pp 95–103
58. Sousa KS, Vanderdonckt J, Henderson-Sellers B, Gonzalez-Perez C (2012) Evaluating a graphical notation for modelling software development methodologies. J Vis Lang Comput 23(4):195–212
59. Souza VES, Lapouchnian A, Robinson WN, Mylopoulos J (2013) Awareness requirements. In: de Lemos R, Giese H, Mueller HA, Shaw M (eds) Software engineering for self-adaptive systems II. LNCS, vol 7475. Springer, pp 133–161
60. Susi A, Perini A, Giorgini P, Mylopoulos J (2005) The tropos metamodel and its use. Informatica (Slovenia) 29(4):401–408
61. Tomkins CC (2006) An introduction to non-parametric statistics for health scientists. Univ Alta Health Sci J 3(1):20–26
62. van Lamsweerde A (2001) Goal-oriented requirements engineering: a guided tour. In: RE, p 249
63. van Lamsweerde A, Letier E (2000) Handling obstacles in goal-oriented requirements engineering. IEEE Trans Softw Eng 26(10):978–1005
64. van Riemsdijk B, Dastani M, Winikoff M (2008) Goals in agent systems: a unifying framework. In: Proceedings of the 7th international conference on autonomous agents and multiagent systems (AAMAS'08), pp 713–720
65. Wampold B, Drew C (1990) Theory and application of statistics. McGraw-Hill, New York
66. Weyns D, Omicini A, Odell J (2007) Environment as a first class abstraction in multiagent systems. Auton Agents Multi-Agent Syst 14(1):5–30
67. Whittle J, Sawyer P, Bencomo N, Cheng BHC, Bruel J (2010) RELAX: a language to address uncertainty in self-adaptive systems requirement. Requir Eng 15(2):177–196
68. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2000) Experimentation in software engineering: an introduction. Kluwer, Norwell
69. Yang Z, Li Z, Jin Z, Chen Y (2014) A systematic literature review of requirements modeling and analysis for self-adaptive systems. In: Requirements Engineering: Foundation for Software Quality REFSQ 2014. Essen, pp 55–71
70. Yu E (1995) Modelling strategic relationships for process reengineering. PhD thesis, University of Toronto, Department of Computer Science
71. Yu E (2009) Social modeling and i*. In: Borgida AT, Chaudhri V, Giorgini P, Yu E (eds) Conceptual modeling: foundations and applications. Lecture notes in computer science, vol 5600. Springer, Berlin, pp 99–121
72. Zhu Q, Lin L, Kienle HM, Müller HA (2008) Characterizing maintainability concerns in autonomic element design. In: ICSM, pp 197–206