Auton Agent Multi-Agent Syst (2017) 31:423–468
DOI 10.1007/s10458-016-9327-7

CrossMark

Requirements specification via activity diagrams for agent-based systems

Yoosef Abushark[1] · Tim Miller[2] · John Thangarajah[1] · Michael Winikoff[3] · James Harland[1]

Published online: 9 February 2016
© The Author(s) 2016

Abstract Goal-oriented agent systems are increasingly popular for developing complex applications that operate in highly dynamic environments. As with any software these systems have to be designed starting with the specification of system requirements. In this paper, we extend a popular agent design methodology, Prometheus, and improve the understandability and maintainability of requirements by automatically generating UML activity diagrams from existing requirements models; namely scenarios and goal hierarchies. This approach aims to overcome some of the ambiguity present in the current requirements specification in Prometheus and provide more structure for representing variations. Even though our approach is grounded in Prometheus, it can be generalised to all the methodologies that support similar notions in specifying requirements (i.e. notions of goals and scenarios). We present our approach and an evaluation based on user experiments. The evaluation showed that the activity diagram based approach enhances people's understanding of the requirements, makes it easier to modify requirements, and easier to check them against the detailed design of the agents for coverage.

Keywords Requirements specification · AOSE · Prometheus methodology

✉ Yoosef Abushark
yoosef.abushark@rmit.edu.au

Tim Miller
tmiller@unimelb.edu.au

John Thangarajah
john.thangarajah@rmit.edu.au

Michael Winikoff
michael.winikoff@otago.ac.nz

James Harland
james.harland@rmit.edu.au

1 RMIT University, Melbourne, Australia

2 University of Melbourne, Melbourne, Australia

3 University of Otago, Dunedin, New Zealand

1 Introduction

Intelligent agent technology is increasingly employed in the development of complex applications such as UAVs, military simulation systems, logistics and planning [30,31]. The development of such systems requires appropriate design methodologies. To this end, numerous agent-oriented software engineering (AOSE) methodologies have been proposed [22,44]. A fundamental aspect of all these methodologies is the specification of requirements. While all these methodologies include requirements specifications, they vary in the approaches and techniques applied.

Prometheus [33] is a well-established methodology that provides support for the complete agent software development cycle. Currently, the requirements of the system are specified via *scenarios*, *goals* and *interfaces* to the environment. A scenario is similar to a use case [24] and describes a particular run of the system as a sequence of steps. These step types include percepts,[1] actions or goals. Goals can be decomposed into sub-goals, using a goal-tree. The combination of the scenarios together with the goal trees forms part of the requirements for the system.

There are a number of limitations in the current representation: (i) scenarios only capture a sequence of steps, which means that steps that could be performed in parallel can only be shown as a sequence; (ii) variations to the scenario are captured informally as English text; and (iii) the distribution of the requirements between scenarios and goal-trees means that it is not always easy to understand and modify the requirements specification. This makes it more difficult to check for coverage of the requirements, and can lead to potential design errors (see Sect. 5).

In this work, we propose using UML *activity diagrams* as a more structured representation of requirements specification to complement the process in the Prometheus agent design methodology. Scenarios and goal-trees have their own advantages, and form an integral part of the Prometheus methodology. Hence our proposal is not to replace these design artefacts but to *complement* them with activity diagrams as a way of overcoming the current limitations. We do this by taking a scenario and the corresponding goal-trees[2] and generating an equivalent activity diagram. The activity diagram generated represents and models the flows of only one scenario at time. The designer can use this activity diagram to: (i) illustrate which steps in the scenario can be attempted in parallel or specify that the ordering does not matter; (ii) specify variations to the scenario in a structured manner; and (iii) better understand and modify the requirements, due to having a holistic view of the requirements specification in one diagram.

We performed a series of evaluations on fifteen participants, who were tasked with interpreting and modifying two different sets of requirements—one with an activity diagram and one without—measuring their performance and asking for qualitative feedback. Our results demonstrate that the participants were able to complete the tasks more correctly and faster using an activity diagram, and that they unanimously preferred the addition of activity diagrams. Even though our approach is grounded in Prometheus, it can be generalised to all the methodologies that support similar notion in specifying requirements (i.e. notions of goals and scenarios).

This paper is organised as follows. Section 2 briefly introduces required background prior to presenting our activity diagram based approach in Sect. 3. We discuss the potential advantages of the approach in Sect. 4, and an empirical evaluation in Sect. 5. Finally, Sect. 6 outlines some related work, and Sect. 7 concludes.

[1] Events that represent inputs to the system from the environment.

[2] That is, the goal-trees involving the scenario's goal steps.

2 Background

In the context of this article, an *agent system* is a *goal-oriented computer program* that acts on behalf of the user to achieve the goals desired. Such systems must be autonomous in making the most suitable decisions based on their current environmental situation. The level of abstraction that is offered by this paradigm helps system designers to manage the complexity, and hence improve the construction of complex systems [25]. In addition, these systems exhibit a set of characteristics that make this paradigm suitable for building systems that operate in a highly dynamic and, often, unpredictable environments.

This section focuses on briefly explaining the related topics to this article. We first discuss some of the most commonly used agent-oriented software engineering methodologies. This involves requirements specifications as part of the process. The next section provides a brief introduction to goal-oriented requirements engineering. We then explain the UML activity diagrams and the adopted notations in our approach.

2.1 AOSE methodologies and requirements specifications

The agent-oriented software engineering field has a number of methodologies that assist developers in structuring, planning and controlling the development process, including MaSE [11], ROADMAP [26], Tropos [7], INGENIAS [35], Gaia [45,49], PASSI [8] and Prometheus [33]. Also, there are a number of frameworks that allows process engineers to create custom agent-oriented methodologies, such as FAML [4] and O-MaSE [14]. Each AOSE methodology consists of a number of phases where a number of activities take place. The specification of requirements is usually the initial activity performed in the development lifecycle. Whilst our method can be applied to a number of these methodologies, in this article we focus on Prometheus, both due to our greater familiarity with this system and in order to provide specific examples of our technique. However, we briefly explain some other methodologies to show that they share similar design notions with respect to requirements specifications.

The Prometheus Methodology

The Prometheus methodology consists of three phases: the system specification phase, the architectural design phase and the detailed design phase.

We now briefly introduce the relevant parts of Prometheus, using a trading agent system as a running example.[3] This system models the processes that take place in a sales transaction, and includes three agents: the seller, the buyer and the banker. The seller agent must send the list of products to the buyer agent when it receives the "store opening" percept. The buyer agent then selects a product. After that, the seller agent should send the buyer the price of the selected item. The buyer agent should then proceed with the payment through the banker agent. The banker agent then processes the payment and notifies both the seller and the buyer about the payment process outcomes (approved or denied). The order of these notifications is not important (e.g. seller first and buyer second or the other way around). In the case of an approved payment, the seller must send the item to the buyer.

In the system specification phase (Fig. 1), a translation of the problem that the intended system needs to solve is done based on the user requirements. Briefly, the requirements are taken as an input and the initial specification of the system drawn defining the goals and the scenarios. Additionally, the external entities (actors), system inputs (percepts) and system outputs (actions) of the intended system are defined. The primary outputs from this phase

[3] The description here is very brief, and we refer the reader to the literature for a full description [33].

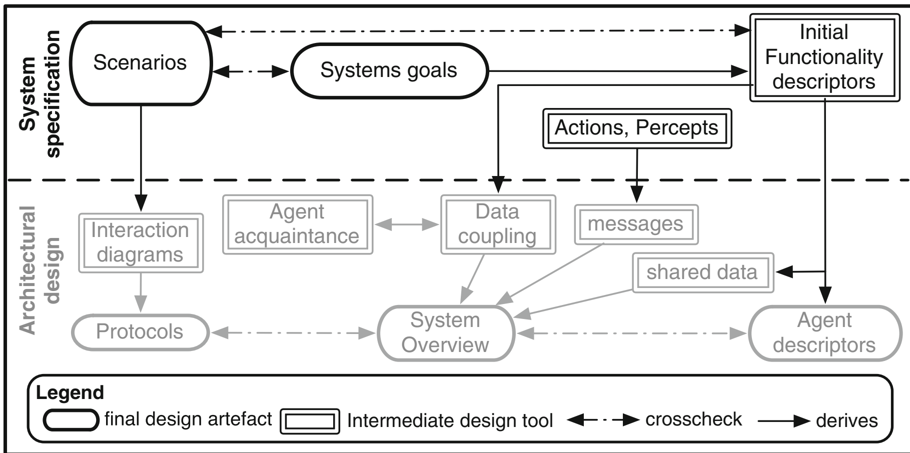


Fig. 1 Phases of the Prometheus methodology

Fig. 2 Sale transaction scenario description

	Type	Name	Role
1	Percept	Store_Opening	Seller
2	Goal	Send_Item_List	Seller
3	Goal	Select_Item	Buyer
4	Goal	Send_Item_Price	Seller
5	Goal	Make_Payment	Buyer
6	Goal	Validate_Card	Banker
7	Goal	Notify_Participants	Banker
8	Goal	Send_Item	Seller

are a goal overview diagram, definition of the interface between the system-to-be and its environment, and a collection of scenarios (see Fig. 1). In the Prometheus methodology, scenarios consist of a sequence of steps, where each step can be an action (i.e. something the agent does), a percept (i.e. an input from the environment), a goal to achieve, or a sub-scenario. Each step is associated with a number of roles. Figure 2 shows a scenario in the trading agent system. Note that the aim of the scenario is to capture an example trace through the system's behaviour, and it therefore does not specify a complete set of execution traces. However, as we shall see in Sect. 3, we can use the information in scenarios and goal overview diagrams to construct constraints that must be met by the detailed design of a multi-agent system designed to meet these requirements. Goals are commonly modelled using a *goal diagram* that shows the relationship between goals, including how goals are decomposed into sub-goals. There are three types of goal decompositions (only two are shown in Fig. 3): disjunctive, undirected conjunctive or directed conjunctive. The disjunctive decomposition (denoted by *OR*) implies that a parent goal is realised if any of its children is realised. The undirected conjunctive decomposition (denoted by *AND*) implies that a parent goal is realised if all its children are realised in some, unspecified, order. The directed conjunctive decomposition (denoted by *AND* with dashed arrows between the children) implies that a parent goal is realised if all its children are realised in the specified order. The dashed arrows between the children indicates the ordering constraints (e.g. *Validate Card* before *Notify Participants*).

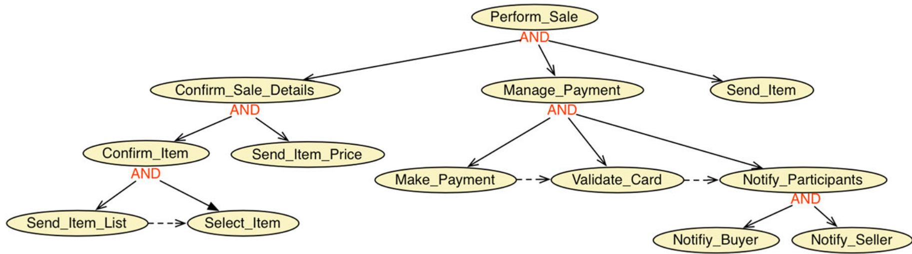


Fig. 3 Goal overview diagram for the trading agent system

Table 1 Notions adopted in specifying requirements in the seven AOSE methodologies

	Goals	Use cases/scenarios	Other notations and models
Prometheus	✔	✔	Role overview diagram
Tropos	✔	✘	Actor diagram and rationale diagram
MaSE	✔	✔	Sequence diagrams and concurrent task model
INGENIAS	✔	✔	Organisation model and task model
Gaia	✘	✘	Environmental, role, protocol and interaction models
ROADMAP	✔	✔	Environmental, knowledge, role, protocol and interaction models
PASSI	✘	✔	UML packages, sequence diagrams and activity diagrams

To distinguish between the two ANDs, we refer to the *directed*-conjunctive with *SEQ* in this article. Figure 3 shows a goal overview diagram for the trading agent system,[4] which outlines the goals and sub-goals required to successfully achieve a sales transaction.

Other AOSE methodologies and requirements specifications

AOSE methodologies provide the necessary framework to organise the development activities and guide designers throughout the development life-cycle. We briefly explain six methodologies other than Prometheus to show that they share similar notions in specifying requirements. This is not intended to serve as a comprehensive list of methodologies.[5] As Table 1 shows, all the methodologies except for Gaia capture requirements using artefacts that include a goal hierarchy and/or activities to be executed by agents (scenarios).

These six methodologies were chosen based on the approach adopted in [9]. The authors in [9] adopted a *multi-stage* selection approach, in which the set of methodologies is reduced through applying the following three criteria:

– *Documentation* the selected methodology should be well established and described in detail.
– *Tool support* a methodology that is supported by a computer aided software engineering tool has more value than the one without.

[4] The provided goal overview is a possible design and it does not necessarily represent a good one.

[5] We refer the reader to [18] for a more comprehensive comparison between AOSE methodologies.

– *Maturity* the selected methodology should be well recognised by the agent community, and has been continuously improved.

In [9] the authors stated that seven methodologies out of the AOSE methodologies in the literature meet these three criteria. We include all of them, but ADEM [41, Sect. 7.2], since it provides a framework rather than a methodology. Also, we include ROADMAP even though it is weak on the maturity criterion, since it represents an extension to Gaia, and supports similar notions like Prometheus.

Tropos has five phases [7]: early requirements, late requirements, architectural design, detailed design and implementation phase. In the early requirement phase, the system's stakeholders are identified as well as the systems objectives, which leads to the *systems' actors and goals*. Then, the obligation of the system towards its environment is determined as the main activity of the late requirement phase. Tropos adopts the *i** organisational modelling framework in modelling the requirements [7], since the *i** framework supports the notions of goals, actors and their dependencies [46]. Even though Tropos does not have the notion of scenarios as part of requirements specifications process, the use of use-cases has been proposed in its extended version the *secure Tropos* methodology [3].

The INGENIAS methodology enables agent designers to develop an agent-based system through its five *viewpoints* [35]: organisation, agent, goals/tasks, interactions, and environment. The methodology, through its viewpoints, promotes a number of abstraction concepts, such as agent, goal and mental states. The requirements specification process takes place in the first three viewpoints. In the organisational view point, the *goals* of the intended system are defined. Also, the *tasks* which the agents need to execute to achieve the desired goals are specified in this viewpoint. In the goals/tasks viewpoint, the goals and tasks are decomposed and refined further. The tasks are to be defined in terms of what, why, input, output and goals to be achieved and affected. Given that, we can claim that tasks are similar to scenarios.

The Multi-Agent System Engineering (MaSE) methodology allows the development of agent-based systems through two main phases: analysis and design [11]. Each phase has its own steps that result in different artefacts. In the context of this article, we are concerned with the analysis phase, where the system goals and roles are identified. This phase has three steps: (1) capturing goals, where the *goal hierarchy* is generated, (2) applying use cases, where the *use-cases and sequence diagrams* are constructed; and (3) refining roles, where the concurrent tasks and roles are modelled. As a first step, the designer needs to identify what the intended system wants to achieve (*system's goal*). Then, the system's roles and their tasks are specified in a form of *use-cases* that define the desired behaviour. In fact, *use-cases* are elicited from the context of the intended system into *positive* and *negative* use-cases. The positive use-cases state the normal behaviour of the system, whilst the negative ones describe the broken or erroneous behaviours. These *use-cases* are then converted into *sequence diagrams* to visualise the flow of events between different roles. Finally, a transformation of the *goal hierarchy* and the *use-cases*, via sequence diagrams, into roles and their associated tasks takes place. Similar to Prometheus, MaSE uses a goal hierarchy to model the goals of the intended system. In MaSE use-cases are specified using sequence diagrams that capture the flow of steps, which are richer representation of scenarios in Prometheus.

The Gaia methodology [45,49] models requirements by using four models [49]: an environmental model, preliminary role and interaction model, and organisational rules. We are concerned with the second model in this article (role models). Roles are identified through two types of attributes: (1) their permissions and rights; and (2) their responsibilities or

functionality. Responsibilities are similar to scenarios, as they focus on the functions to be executed by agents.

The ROADMAP methodology extends the Gaia methodology for developing open systems [26]. One of Gaia's weaknesses is the lack of support of the goal notion (e.g. social goals of the intended system) in the modelling process. The ROADMAP methodology, on the other hands, overcomes this issue by introducing a number of extensions. First, it introduces an initial phase that concerns the requirement elicitation process via *use-cases*. Second, the methodology includes the definition of the local social *goals* as part of the role modelling process.

PASSI (Process for Agent Societies Specifications and Implementation) is a step-by-step methodology that guides designers throughout the development life-cycle. The methodology has five process component referred to as models: (1) system requirements model, (2) agent society model, (3) agent implementation model, (4) code model and (5) deployment model. Each model is composed of a number of phases [8]. In this work we concerned with system requirements model. This model consists of four phases: (1) domain requirement description, (2) agent identification, (3) role identification and (4) task specification [8]. In the domain requirement description the functionalities of the system are identified through using *use case* diagrams. The responsibility of each agent is attributed in the agent identification as UML packages. Then, the responsibilities of each agent based on a role-specific scenarios are explored via sequence diagrams. In the task specification phase activity diagrams are used in specifying the capabilities of each agent.

As we have seen, in the agent-oriented software engineering methodologies discussed, requirements specifications generally include [44, Sect. 4] scenarios, which are instances of the desired execution behaviour, and goals, which are intended states of the system. Note that that idea of using goals (and scenarios) is not unique to AOSE, but has also been used more broadly in requirements engineering [10,42,46], which we discuss in the next section.

2.2 Goal-oriented requirements engineering

A fundamental aspect of any software engineering methodology is the specification of requirements and the related area of *requirement engineering* (RE). Requirements engineering is a process that provides systematic techniques to ensure the quality of the system requirements [37, p. 5]. This process encompasses a number of activities including: requirements elicitation, analysis, specification, verification and management. Many requirements engineering methodologies have been proposed to organise the activities of the process. While these methodologies are common in their inclusion of the analysis and design activities, they may vary in the approaches and techniques applied [28].

In goal-oriented RE, the intended system is defined and analysed in terms of goals, which are the objectives the system must attain [43, p. 259]. These goals are classified, based on the type of concerns, into two categories [42]: (1) functional goals that concern the services provided by the system; and (2) non-functional goals that specify the quality of these services, such as the reliability of the system. These goals can be modelled through the following [42,43]: (i) their types (e.g. functional and non-functional), (ii) attributes, such as the goals priority; and (iii) their associations to other entities in the model including the inter-goals links. The main purpose behind these modelling techniques is to facilitate reasoning about goals [42]. In the context of agent-oriented paradigm, most of the methodologies we discussed earlier (all but Gaia) use goals in specifying the requirements [12].

2.3 UML activity diagrams

Activity diagrams are a type of behavioural model that describes the dynamic aspects of a given system [32]. They visualise processes (sequence of steps) and model the work- and data- flows of the system. UML provides designers with a range of graphical notations for modelling activity diagrams including activity nodes and activity edges. The creation of these diagrams must conform with the the UML activities metamodel to ensure their semantics. In fact, this metamodel provides an enormous number of entities to enhance the semantics of activity diagrams. Figure 4 shows a subset of the activity diagram metamodel that defines the entities considered in our approach described in Sect. 3. As Fig. 4 illustrates, we adopt six entities out of the UML activity diagram notations in our approach as follows (Fig. 5):

(1) *Action Node* the fundamental atomic node in the activity diagrams (Fig. 5a). It is notated by a rectangular shape with rounded edges.
(2) *Regular Activity Edge* a directed connection that shows the control flow between the different actions within an activity diagram (Fig. 5b). These edges can optionally have conditions (*guards*) to constrain their flow.
(3) *Initial Node* a control node that initiates the execution of an activity/scenario (Fig. 5c). We restrict the activity diagrams to have only one initial node.
(4) *Final Node* a control node that shows the end of an activity/scenario (Fig. 5d).
(5) *Fork/Join* a control node that splits the execution flow into multiple threads to be executed independently, and then synchronises them (Fig. 5e).
(6) *Decision/Merge* a control node that splits the execution of an activity into multiple alternate flows with only one flow to be executed, based on the *guards* on the outgoing activity edges from the decision point (Fig. 5f).

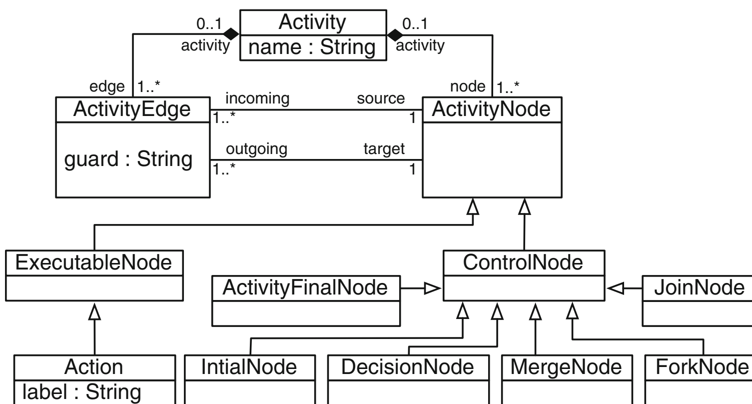


Fig. 4 Activity diagrams metamodel

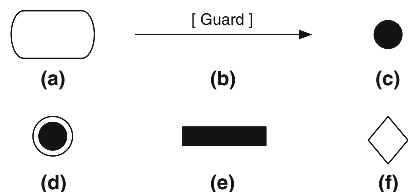


Fig. 5 Adopted UML activity diagram notations **a** action node **b** regular activity edge **c** initial node **d** final node **e** fork/join **f** decision/merge

3 Method

In this section, we present our approach for automatically constructing an activity diagram from a scenario and goal overview diagram, and briefly discuss a prototype implementation. We use the trading agent described in Sect. 2 as a running example.

Our approach aims to provide agent-based software designers with an activity diagram that complements the scenario and goal overview diagram by modelling the possible paths in a given scenario, with consideration of information from the goal overview diagram relevant to that scenario. The activity diagram includes alternatives of the goal steps in the scenario according to the goal overview diagram.

Although a scenario is a single sequence of steps there may be different ways to realise the same scenario. This is because when there are goal steps, the goals may also be realised (and hence, the requirement specified) through its children from the goal overview diagram. For example, the goal step *"Notify Participants"* in Fig. 2 could be implemented through the step itself, the step with its children, or just the children, (*"Notify Buyer"* and *"Notify Seller"*, see Fig. 3).

The construction process of the intended activity diagram involves two phases:

(1) *Step-wise activity diagram generation* this phase takes one step—of a given scenario—at a time, and construct its equivalent activity diagram structure. Then, it concatenates the different structures to form the complete activity diagram corresponding to the specified scenario.
(2) *Activity diagram reduction* the generation phase results in an activity diagram with duplicate nodes. This phase intends to reduce these duplicates, if possible, while preserving the semantic of the original activity diagram.

3.1 Step-wise activity diagram generation phase

The purpose of this phase is to construct an activity diagram from the specified scenario by combining it with the the relevant information from the goal overview diagram to each goal step, if any. Such a description reflects the transformation of the steps in a given scenario into activity diagram control fragments.

Since the proposed approach is grounded in the Prometheus methodology, it assumes scenarios include four types of steps: action, percept, goal, and sub-scenario. We transform every step into a control fragment based on the type of the step. For the first two types (actions and percepts), they are transformed into *sequential* control fragments. The goal steps are transformed into combination of *alternative*, *parallel*, and *sequential* control fragments, depending on the decomposition of these goal steps in the goal overview diagram. We flatten sub-scenario steps by including their steps in the main scenario.

Each control fragment in the activity diagram includes action nodes that represent the steps in the scenario considered. Note that usually action nodes in activity diagrams model the execution of behaviour, such as operation invocations. However, we are using activity diagrams as part of requirements specification, and so an activity diagram with a certain sequence of steps represents a requirement that the subsequently designed system must fulfil. Specifically, where a goal step in a scenario appears as a corresponding action node in an activity diagram, it does not denote the execution or achievement of the corresponding goal, but a requirement that the designed system be able to achieve the goal. This distinction is important because the design process may end up refining the goals. For example, consider a scenario S that has a single step, $Goal1$. This yields an activity diagram with one action node

in a sequential control fragment. This does not mean that $Goal1$ is executed, but that the design needs to be able to achieve $Goal1$. In the case that $Goal1$ has child goals, a designer may choose to design a system that does not implement $Goal1$ directly, but rather, achieves its children.

Therefore, in mapping the goal steps in a given scenario to an activity diagram we take into account the goal hierarchy (as depicted in the goal overview diagram). The reason is that the process of designing a system relative to a given scenario may focus on parent or children goals of the scenario goal steps. For example, step 7 in the Sale Transaction Scenario (Fig. 2) is *Notify_Participants*. It is possible that the design realises this step by adopting the goal itself. However, it is also possible that the design refines the scenario into more detail, and that instead of adopting the goal *Notify_Participants*, instead its two children *Notify_Buyer* and *Notify_Sender* are adopted, as such adoption will realise the parent goal (*Notify_Participants*) as desired.

This means that when mapping a scenario to an activity diagram, a goal step may be mapped to a process that combines the goal with its parent or children. In developing the rules for this mapping we follow the underlying principle that *the scenario specifies design decisions, and that these decisions must be honoured*.

For example, consider a variant scenario that replaced *Notify_Participants* with *Notify_Buyer* followed by *Notify_Seller*. The scenario specifies a specific order on these two goals. If we allow the design process to replace these two goals with their parent then subsequent refinement might re-introduce the two sub-goals, but without the ordering constraint specified by the scenario. This fails to be consistent with the scenario's constraint on the order.

Another example is where a goal G has two children G_1 and G_2 that are OR-refined. In this case, if we replace a scenario step G_1 with G, then we are losing the information that the scenario designer chose to use G_1 rather than G_2.

We now proceed to define the rules for mapping the goal steps in a scenario to an activity diagram, being guided by the above principle.

Goal steps merging rules

The reason for this merging process is to provide designers with various ways to realise goal steps in their design, by considering the information from the goal hierarchy that is relevant to these goal steps. It depends on the level of abstraction a designer wants their design to capture. For example, a designer may opt to realise a goal step through the goal itself. Or, if the designer considers the goal step in a scenario to be too low-level and detailed, then that may realise the step in terms of its parent goal. On the other hand, if the designer considers the goal step to be too high-level, then they may refine the goal, and realise it in their design in terms of the goal step's descendants.

Recall that we consider a goal in an activity diagram to represent a requirement that must be realised in the subsequent design, rather than the achievement of the goal. This means that the presence of a goal's children does not subsume the goal itself. For example, consider a goal G with three AND-refined children, G_1, G_2, and G_3. One possible design would have a plan for achieving G that makes use of sub-goals G_1 to G_3, each with their own plan.

Let us first consider the possibility of mapping a goal step G in a given scenario in terms of its descendants from the goal overview diagram. If instead of designing (and adopting) G, we design its children from the goal overview diagram, then the constraints that are captured by the scenario are not violated. To see this we consider three cases, corresponding to the decomposition type in the goal overview diagram. There are three different decompositions captured by the goal overview diagram (OR, SEQ, AND):

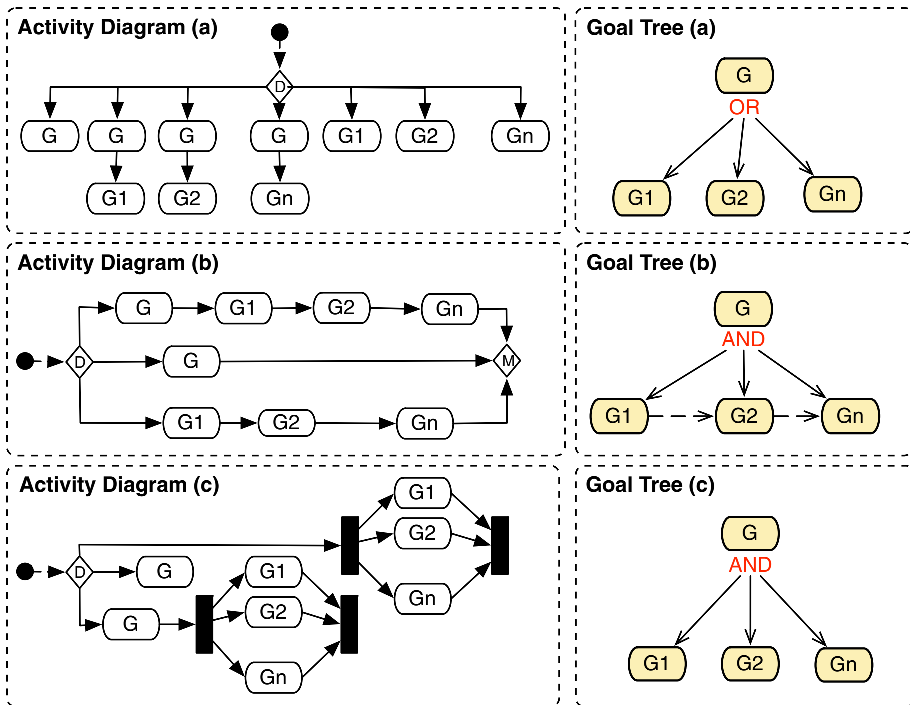


Fig. 6 Activity diagram control fragments equivalent to the goal step transformation

(1) *Disjunctive decomposition (OR)* If a goal step G has, in the goal hierarchy, as children G_1 OR G_2, we can either end up realising this step through designing a design that can adopt either G_1 or G_2 (perhaps deciding on which at run-time), or we can make a design time decision to select one of them and, say, only design G_2. Either option is fine: if either G_1 or G_2 is adopted, then G is known to be realised as desired. In this case, the transformation is to an alternative fragment between:[6] the goal step G; the goal step G with one of its children in sequence, for each child; and each of its children (refer to Fig. 6a).

(2) *Directed-conjunctive decomposition (SEQ)* If a goal step G has, in the goal hierarchy, as children G_1 SEQ G_2, then adopting G_1 *followed by* G_2 will realise G as desired. In this case, the transformation is to an alternative fragment between: the goal step G; the goal step G with all of its children in the sequence specified; and all of its children in the sequence specified (refer to Fig. 6b).

(3) *Undirected-conjunctive decomposition (AND)* If a goal step G has, in the goal hierarchy, as children G_1 AND G_2, then a detailed design that adopts the children (either in parallel, or in a specified order) will realise G, as required. In this case, the transformation is to an alternative fragment between: the goal step G; the goal step G followed by all of its children as a parallel fragment; and all of its children as a parallel fragment (refer to Fig. 6c).

[6] We include the option of having both the goal as well as its children because in the case where the design targets a BDI platform one typically posts the parent goal, which leads to the posting of the children (for an OR, one of the children) goal, so both parent and children are possible.

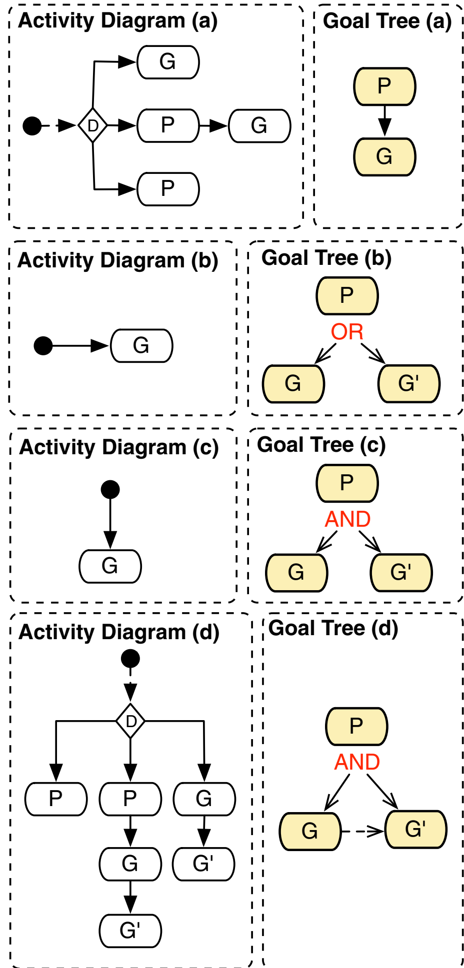


Fig. 7 Mapping a goal step in terms of its parent

We now consider the possibility of mapping a goal in terms of its parent. We consider four cases: where the goal G being mapped is an "only child" (i.e. its parent P satisfies $children(P) = \langle G \rangle$), and where it has siblings, in which case there are three cases, corresponding to the decomposition of P in the goal overview diagram (OR, AND, SEQ). We now consider the four cases.

Case 1: G is an only child—In this case if we replace G with P and the detailed design subsequently refines this, then because P only has G as child, the only possible refinement is back to G, which is consistent with the scenario. So it is safe to replace G with P in this case: subsequent refinement can not be inconsistent with the scenario (Fig. 7a).
Case 2: G has OR siblings—In this case if we replace the step G with P then subsequent refinement of P at the detailed design could choose to replace G with one of its siblings. This is inconsistent with the scenario, since adopting P implies the adoption of one of the children, which may not necessarily the one specified in the scenario. Thus, we cannot replace G with P in this case (Fig. 7b).

Case 3: G has AND siblings—Suppose that G has a parent P and a single sibling G' and that the scenario includes G as well as G'. Suppose we replace "G and G'" with P. This creates a problem because the scenario specifies an ordering: G before G'. however, if we replace G and G' with P, then subsequent refinement may end up violating the order specified in the scenario. So in general, we cannot replace "G and G'" with P in this case (Fig. 7c). Note that the order specified in the scenario may not actually be significant: it may only be there because the scenario must specify an order. However, the point is that we do not, and cannot, know whether the order of G and G' is significant or not.

Case 4: G has SEQ siblings—Suppose that G has a parent P which is SEQ- decomposed into G followed by G'. As per case 3, we assume that the scenario includes both G and G' as steps, and furthermore, that G and G' appear in the scenario in the order that is consistent with their SEQ decomposition. If we replace G and G' with P then any subsequent refinement would re-introduce G and G' in the correct order, and so will be consistent with the scenario. In this case, we can allow for mapping a goal step G in terms of its parent under a number of conditions: that the scenario includes all its siblings, that they appear contiguously in the correct order (with respect to the goal overview diagram), and that we map all the children of P as an option to design P instead (Fig. 7d).

Since our approach processes the scenario one step at a time, we address this case by simply pre-processing the scenario, finding consecutive goals in the scenario that are SEQ siblings in the goal overview diagram, and replacing them in the scenario with their parent.

So, to summarise, a goal step in the scenario is a non-leaf goal (a parent goal) in the goal overview diagram, the scenario should, in some cases, include the children as *alternatives*, as such a step can be realised through the children in the detailed designs. Similarly, under certain situations, a goal step can be realised in terms of its parent goal.

We note here that, whilst there are subtle variations to these cases that do indeed comply with the scenario, our aim is not to provide the designer with all possible cases, but rather present some intuitive variations that they can work with.

3.2 Formalising the generation approach

We now formalise the merge and the transformation process of the goal steps in a given scenario, since the transformation of other steps is straightforward as discussed in Sect. 3.1. A scenario comprising steps $S_1 \ldots S_n$ is translated to a sequence of activity diagram fragments, where each step is the translation of the corresponding S_i. Action and percept steps are transformed into sequential fragments. However, goal steps are transformed into different fragments based on the different decompositions in the goal overview diagram. Let G be the goal corresponding to the goal step being mapped, let P denote its parent, $children(G)$ denote its children as a sequence of labels (where the order is the order of execution for a directed-AND decomposition, and is arbitrary otherwise), and let $decompose(G)$ denote the decomposition type of the children of G, i.e., one of the following:

(1) *Disjunctive decomposition (OR)* G has children $G_1 \; OR \cdots OR \; G_n$ (Fig. 8a).
(2) *Directed-conjunctive decomposition (SEQ)* G has children $G_1 AND \rightarrow \cdots \; AND \; \rightarrow G_n$ (Fig. 8b).
(3) *Undirected-conjunctive decomposition (AND)* G has children $G_1 \; AND \cdots AND \; G_n$ (Fig. 8c).
(4) $Leaf$ G is a $Leaf$ if it has no children (Fig. 8d).

For brevity we also define a simple abstract notation for depicting activity diagram control fragments: a name is short hand for a step and we use $seq(a_1, \ldots, a_n)$ to denote the action

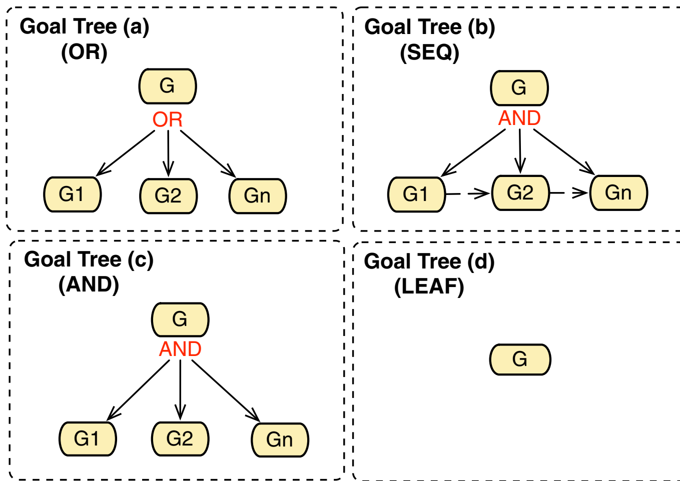


Fig. 8 Goal tree decomposition types

nodes within the activity diagram where the a_i are joined sequentially; $par(a_1, \ldots, a_n)$ to denote the action nodes within the activity diagram where all the a_i are triggered to run in parallel ; and $alt(a_1, \ldots, a_n)$ to denote the action nodes within the activity diagram where there is a decision point: exactly one of the a_i is selected. Each outgoing activity edge from the decision point is guarded to ensure that only one flow is selected. These guards are formed during the generation process based on what is included in each flow. For example, suppose a goal step G being mapped is the only child of its parent P and it has one child G'. This goal step can be realised through its parent *followed by* the goal step G *followed by* the child G', and hence the activity diagram should capture this as one of the flows that realise G. To restrict the selection of this flow, a guard like $[Flow = Parent\ .\ Step\ .Child]$ should be placed on the outgoing activity edge from the decision point that belongs to that flow. It is worth noting that activity diagrams are meant to be processed by designers (human beings) and not by machines. Thus, guards do not need to be in formal notation.

We transform a scenario S, consisting of steps named $S_1 \ldots S_n$ to the activity diagram description denoted by $seq(\widehat{S_1}, \ldots, \widehat{S_n})$. We use S_i to denote a step in a scenario, and $\widehat{S_i}$ to denote the corresponding action node in the activity diagram.

First, we pre-process the scenario and substitute any SEQ decomposition siblings that appear in the scenario with their parents, as discussed in Case 4 above:

$$seq(S_1, G_i, \ldots G_j, S_n) = \begin{cases} seq(S_1, G, S_n) & \text{if } children(G) = \langle G_i, \ldots, G_j \rangle \\ & \wedge\ decompose(G) = SEQ \\ seq(S_1, G_i, \ldots G_j, S_n) & \text{otherwise} \end{cases}$$

in which S_1 and S_n are (possibly empty) sequences of scenario steps.

Next, we analyse each step and in some cases, replace goals with parents or children. If S_i is the name of a goal step then $\widehat{S_i}$ (i.e. $\widehat{G}$) depends on the decomposition type of G, formalised as:

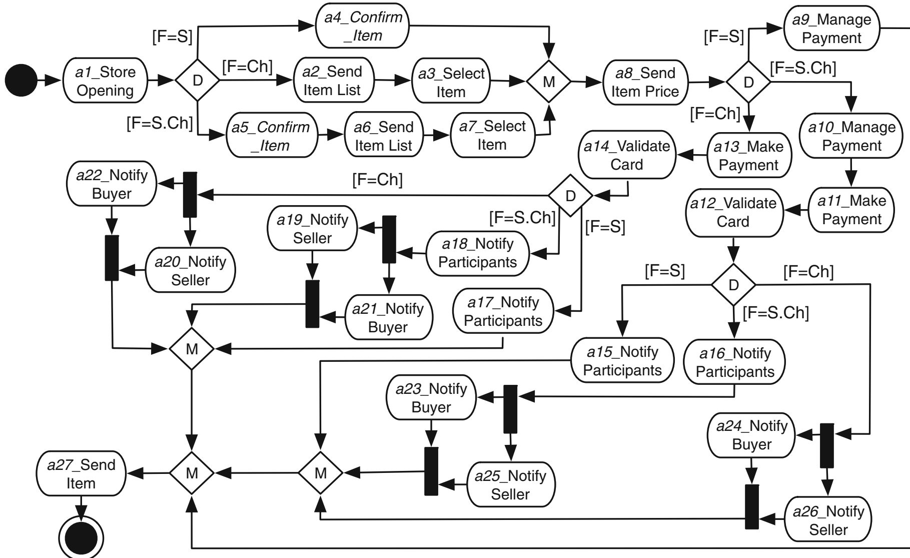


Fig. 9 Activity diagram that merges the scenario in Fig. 2 with the goal tree in Fig. 3 (*F* flow, *S* goal step, *Ch* children, *P* parent)

$$\widehat{G} = \begin{cases} seq(\overline{G}) & \text{if } \mathrm{G}\ is\ a\ Leaf \\ alt(\overline{G}, seq(\overline{G}, alt(M)), alt(M)) & \text{if } DG = OR \\ alt(\overline{G}, seq(\overline{G}, par(M)), par(M)) & \text{if } DG = AND \\ alt(\overline{G}, seq(\overline{G}, seq(M)), seq(M)) & \text{if } DG = SEQ \end{cases}$$
$$\text{where } DG = decompose(G)$$
$$\text{and } \langle G_1, \ldots, G_n \rangle = children(G)$$
$$\text{and } M = \widehat{G_1}, \ldots, \widehat{G_n}$$

We define the auxiliary function $\overline{G}$, which merges the goal itself, G, with its parent P, to get the sequence $seq(P, G)$ when it is permissible to do so (see earlier discussion), and all other goals to themselves. The goal step G is to be merged with its parent P if and only if the goal step G has no siblings.

$$\overline{G} = \begin{cases} seq(G, P) & \text{if } children(P) = \langle G \rangle \\ G & \text{otherwise} \end{cases}$$
$$where\ P = parent(G)$$

Applying the merging rules the Trading Agent system (Figs. 2, 3) results in the activity diagram shown in Fig. 9.

As the figure depicts, all the action nodes are prefixed with a unique identifier to ensure that duplicate nodes are with different incoming and outgoing vertices and are not merged. For example, consider the activity diagram in Fig. 10. If node $A2$ is represented as a single node on the printed graph, then the sequence $A1$, $A2$, $A5$ would be permitted, which is not the intended semantics. However, this causes duplicates in the graph, which we discuss further in the following section.

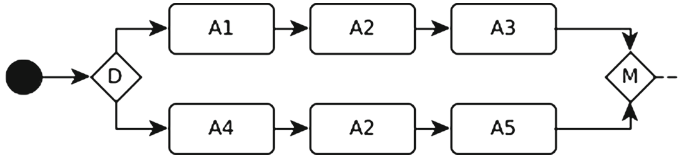


Fig. 10 Example of a non-reducible sub-graph

3.3 Activity diagram reduction phase

As it is shown in Fig. 9, the diagram includes several duplicate nodes, and in some cases, duplicate sub-graphs, which affects its readability. Duplicate nodes are semantically equivalent—that is, they refer to the same event—but are prefixed with unique identifiers; for example, in Fig. 9, nodes *a19_Notify Seller* and *a20_Notify Seller* refer to the same event, but the prefixes are unique. This is done so that the nodes are not merged as one node by the graph drawing tool, which in many cases, would alter the semantics of the activity diagram. The reduction mechanism that we present merges *some* nodes (in fact, some sub-graphs) by removing the prefixes in a sound manner, such that the semantics of the original diagram is maintained.

In this section, we briefly describe how to simplify the original activity diagram generated by the merging rules to improve the readability, while maintaining the same semantics. Our approach is a set of rules to eliminate, if possible, repeated action nodes in the diagram. Thus, we consider such points along with their *merges* in the reduction mechanism.

The following specifies the reduction mechanism as a set of rules on activity diagrams. These rules do not reduce the nodes that have predecessors, successors or both, as that will change the semantics of the original activity diagram. For instance, "A2" node in Fig. 10 can not be reduced. Thus, we make no claims on the completeness or optimality of the approach, however, the rules have worked to simplify the models we have been using them on.

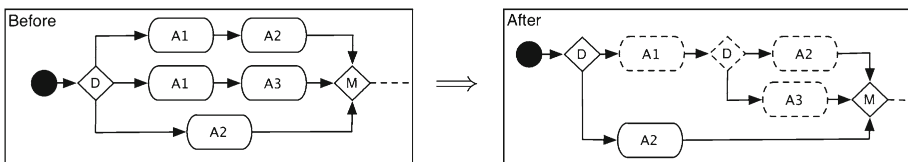
In the following, we use the notation $S_1 \equiv S_2$ to note that two sub-graphs are semantically equivalent; that is, they refer to the same set of events, but have different prefixes on the nodes; for example, *a19_Notify Seller* $\equiv$ *a20_Notify Seller* are unique but semantically-equivalent actions.

Rule 1 *Remove duplicate prefixes in alt: This rule removes duplicate sub-graphs at the start of an* alt *fragment in the activity diagram. A duplicate sub-graph prefix S_2 is removed by merging the sub-graph with its semantic duplicate S_1, and then inserting an alt fragment immediately after S_1. Formally:*

$$\begin{array}{lcl} alt(seq(S_1, S_m), & & alt(seq(S_1, \\ \quad seq(S_2, S_n), & \Longrightarrow & \quad alt(S_m, S_n)), \\ \quad S_r) & & \quad S_r) \end{array}$$

in which $S_1 \equiv S_2$, and S_m, S_n, and S_r are (possibly-empty) graphs.

As an example, consider the following activity diagram with duplicate node $A1$:

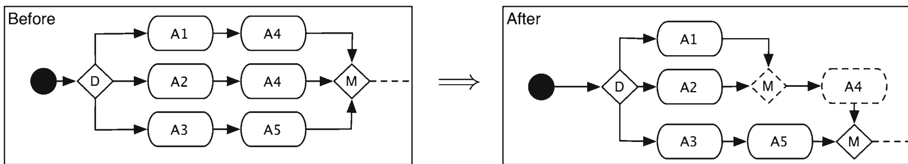


Rule 2 *Remove duplicate suffixes in alt: This rule removes duplicate sub-graphs at the* end *of an* alt *fragment in the activity diagram. It is essentially the reverse of Rule 1. A duplicate sub-graph suffix S_2 is removed by merging the sub-graph with its semantic duplicate S_1, and then inserting an alt fragment immediately before S_1. Formally:*

$$\begin{array}{lcl} alt(seq(S_m, S_1), & & alt(seq(alt(S_m, S_n)), \\ \quad seq(S_n, S_2), & \Longrightarrow & \quad S_1), \\ \quad S_r) & & \quad S_r) \end{array}$$

in which $S_1 \equiv S_2$, and S_m, S_n, and S_r are (possibly-empty) graphs.

As an example, consider the following activity diagram with duplicate node $A4$:



Rule 3 *Remove unnecessary alts: This rule removes* alt *fragments that have only option. Formally:*

$$alt(S_1) \Longrightarrow S_1$$

This rule is useful after an application of Rule 1 in which all children nodes of the decision point are the same. For example, consider the example of Rule 1 above, but in which the third option containing only $A2$ was not in the original activity graph. Applying Rule 1 would result in a graph in which the first decision point had only one option: $A1$. Rule 3 would remove the unnecessary decision point.

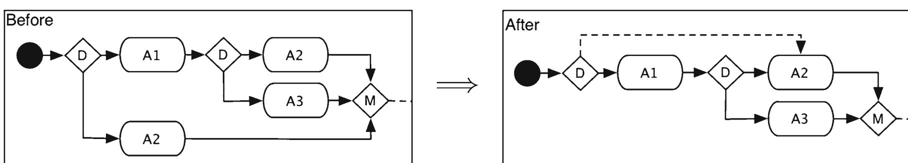
Rule 4 *Remove embedded duplicate sub-graphs: This rule removes duplicate sub-graphs (typically sequences) that occur after a decision point, with the duplicate occurring after a subsequent decision point.*

$$\begin{array}{lcl} alt(seq(S_m, alt(S_1, S_n)), & & alt(seq(S_m, alt(S_1, S_n)), \\ \quad S_2, & \Longrightarrow & \quad S_1, \\ \quad S_r) & & \quad S_r) \end{array}$$

in which $S_1 \equiv S_2$, and S_m, S_n, and S_r are (possibly-empty) graphs.

Note here that the only difference between the left- and right-hand side of this rule is the replacement of S_1 with S_2. While abstractly, there are still two nodes represented, these will be drawn as a single node.

As an example, consider the following activity diagram with duplicate node $A2$:



It may not be immediately obvious as to whether Rule 4 would be so useful. However, these structures occur regularly due to the OR-decomposition rule defined in Sect. 3.1. For example, in the example above, $A1$ is a parent goal and $A2$ and $A3$ are children nodes.

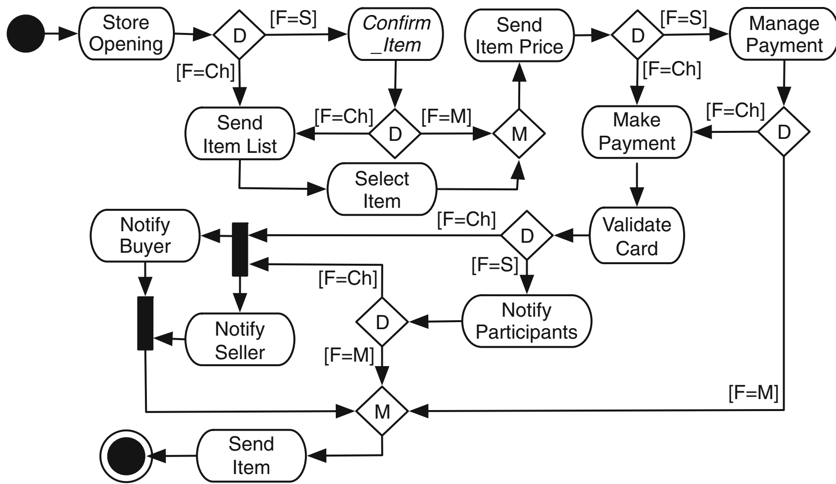


Fig. 11 Refined activity diagram from Fig. 9 (*F* flow, *S* goal step, *Ch* children, *P* parent, *M* merge)

In essence, this simplification is done by identifying repeated sub-graphs, and merging them through applying the aforementioned rules. For example, in Fig. 9 the nodes $a11$ and $a6$ are followed by portions of the graph that are identical to each other in terms of structure, and the labels of nodes (ignoring the numerical prefix on the labels). Figure 11 shows the refined version of the original activity diagram in Fig. 9.

We now prove that the four reduction rules defined above are sound. That is, that they preserve the semantics of the original activity diagram. In proving this we make use of the following notations and concepts.

Given two sequences, $s = \langle s_1, \ldots, s_n \rangle$ and $t = \langle t_1, \ldots, t_m \rangle$ we use $s \cdot t$ to denote the result of concatenating the two sequences, i.e. $s \cdot t = \langle s_1, \ldots, s_n, t_1, \ldots t_m \rangle$. We extend the concatenation operator to also apply to sets of sequences: $S \cdot T = \{s \ \cdot \ t \mid s \in S \wedge t \in T\}$.

We use $[\![S]\!]$ to denote the meaning of S, which is the set of all traces specified by activity diagram diagram S. Specifically, $[\![S]\!]$ is a set of sequences. We assume that S is specified in terms of the constructs $seq(\ldots)$, $par(\ldots)$ and $alt(\ldots)$, and that the semantics satisfies the following two properties:

$$[\![alt(R_1, R_2)]\!] = [\![R_1]\!] \cup [\![R_2]\!]$$
$$[\![seq(R_1, R_2)]\!] = [\![R_1]\!] \ \cdot \ [\![R_2]\!]$$

In other words, the semantics of $alt(\ldots)$ is the union of the semantics of each of the alternatives, and the semantics of $seq(\ldots)$ is the concatenation of the semantics of the components. Since the reduction rules do not mention $par(\ldots)$ we do not need to specify $[\![par(R_1, R_2)]\!]$. We now proceed to prove soundness.

Theorem 1 *Reduction rules 1–4 are sound, i.e. for each rule of the form $L \Rightarrow R$ we have that $[\![L]\!] = [\![R]\!]$.*

Proof For Rule 1, consider the following:

$$
\begin{aligned}
&[\![alt(seq(S_1, S_m), seq(S_2, S_n), S_r)]\!] \\
&\quad = [\![seq(S_1, S_m)]\!] \cup [\![seq(S_2, S_n)]\!] \cup [\![S_r]\!] \\
&\quad = ([\![S_1]\!] \cdot [\![S_m]\!]) \cup ([\![S_2]\!] \cdot [\![S_n]\!]) \cup [\![S_r]\!] \\
&\qquad (\text{since } S_1 \equiv S_2) \\
&\quad = ([\![S_1]\!] \cdot [\![S_m]\!]) \cup ([\![S_1]\!] \cdot [\![S_n]\!]) \cup [\![S_r]\!] \\
&\qquad (\text{since } (S \cdot T) \cup (S \cdot R) = S \cdot (T \cup R)) \\
&\quad = ([\![S_1]\!] \cdot ([\![S_m]\!] \cup [\![S_n]\!])) \cup [\![S_r]\!] \\
&\quad = [\![seq(S_1, alt(S_m, S_n))]\!] \cup [\![S_r]\!] \\
&\quad = [\![alt(seq(S_1, alt(S_m, S_n)), S_r)]\!]
\end{aligned}
$$

Similarly for rule 2:

$$
\begin{aligned}
&[\![alt(seq(S_m, S_1), seq(S_n, S_2), S_r)]\!] \\
&\quad = [\![seq(S_m, S_1)]\!] \cup [\![seq(S_n, S_2)]\!] \cup [\![S_r]\!] \\
&\quad = ([\![S_m]\!] \cdot [\![S_1]\!]) \cup ([\![S_n]\!] \cdot [\![S_2]\!]) \cup [\![S_r]\!] \\
&\qquad (\text{since } S_1 \equiv S_2) \\
&\quad = ([\![S_m]\!] \cdot [\![S_1]\!]) \cup ([\![S_n]\!] \cdot [\![S_1]\!]) \cup [\![S_r]\!] \\
&\qquad (\text{since } (T \cdot S) \cup (R \cdot S) = (T \cup R) \cdot S) \\
&\quad = (([\![S_m]\!] \cup [\![S_n]\!]) \cdot [\![S_1]\!]) \cup [\![S_r]\!] \\
&\quad = [\![seq(alt(S_m, S_n), S_1)]\!] \cup [\![S_r]\!] \\
&\quad = [\![alt(seq(alt(S_m, S_n), S_1), S_r)]\!]
\end{aligned}
$$

which shows soundness for rule 2. For rule 4 soundness is trivial, since the only change is replacing S_2 with S_1 when $S_1 \equiv S_2$. For rule 3 soundness is also trivial. □

3.4 Prototype implementation

We have implemented the mapping from scenario and goal overview diagram to activity diagram as an eclipse plug-in that integrates with the Prometheus Design Tool (PDT). The tool takes the agent design file in an XML format, and tokenises all scenarios and the goal overview diagram out of the design file. The tool applies the merging rules on the specified scenario to generate the abstract description. Then, it uses the abstract description to generate a DOT Graph source script,[7] which can than be used to generate a graphical depiction of the activity diagram (the Graphviz tool does automatic layout of nodes).

4 Potential benefits of the activity-diagrams

The aim of our approach is to provide diagrams that act as coherent structures that merge given scenarios with their related goal decomposition trees. This transformation has a number benefits beside enhancing the way of analysing scenarios, since graphical notations tend to better support tasks that involve comprehension of the overall structure [6,40]. This section

[7] See http://www.graphviz.org/Documentation.php.

discusses some benefits of complementing scenarios and goal overview diagrams with activity diagrams. In fact, the usefulness of supplementing the requirements specification process with activity diagram has been studied in the literature [5,20]. However, in this article we show the benefits of activity diagrams as a complementary artefact to the existing agent oriented software engineering artefacts, which include other notions in addition to the ones existed in the general software engineering.

4.1 Structured representation of scenario steps and their variations

A scenario is a sequence of steps that describe a particular run of the intended system. In some cases, based on the context of a given scenario, there may be a need to go beyond a sequence of steps. Approaches for specifying scenarios in Prometheus and some other agent-oriented methodologies limit the ability to include additional information, such as variations, in a structured way. For example, Prometheus scenarios do not support parallel steps, and variations are specified through natural language description. Using UML activity diagrams provides a more structured way to visualise and specify control fragments in requirements models. Returning to the running example, consider the following variation to the scenario in Fig. 2: "*after the buyer agent receives the item list (step 2) from the seller agent, it may select a product (step 3) or show its disinterest*". Thus, the flow of this scenario should branch after posting the second step ("Send Item") into two choices: (1) posting "Select Item" or (2) posting "Show Disinterest". A natural language description of this may faithfully capture the semantics, but interpreting this in the context of the goal overview diagram is non-trivial, as it requires some mental effort. However, an activity diagram can capture this in a straightforward and unambiguous manner, such as the solution in Fig. 12.

The existing approach (text-based) does not offer a structured way to specify parallel steps. Consider the following requirement from the Trading Agent System: "*The banker agent then processes the payment and notifies both the seller and the buyer about the payment process outcomes (approved or denied)"*. Based on the specification, the banker agent needs to send notification to both buyer and seller agents, but the order of posting these two notifications is not important. In Prometheus, this can be modelled by including a parent goal in the scenario, and then specifying two children nodes with an undirected-AND. However, the variations introduced by these may require some mental effort from a designer to conceptualise. Constructing an activity diagram to represent this through *fork/join* nodes can help in this conceptualisation.

4.2 Understandability

In Prometheus, the specification of the intended system is distributed across two artefacts: scenarios and a goal overview diagram. The merged activity diagram provides a holistic view about how to design a given scenario, and enables a more straightforward understanding of the behaviour of the system in the context of a given scenario. Importantly, the activity diagram explicitly represents the possible alternate sequences of the two models. For example, the activity diagram in Fig. 12 has seven possible execution runs after posting the "Send Item Price" step, which requires more mental effort to derive from the scenario (Fig. 2) and goal overview diagram (Fig. 3). According to the experiments we conducted (Sect. 5.3), participants were quicker in eliciting the possible alternatives for a given step using activity diagram than the existing approach (scenario and goal overview).

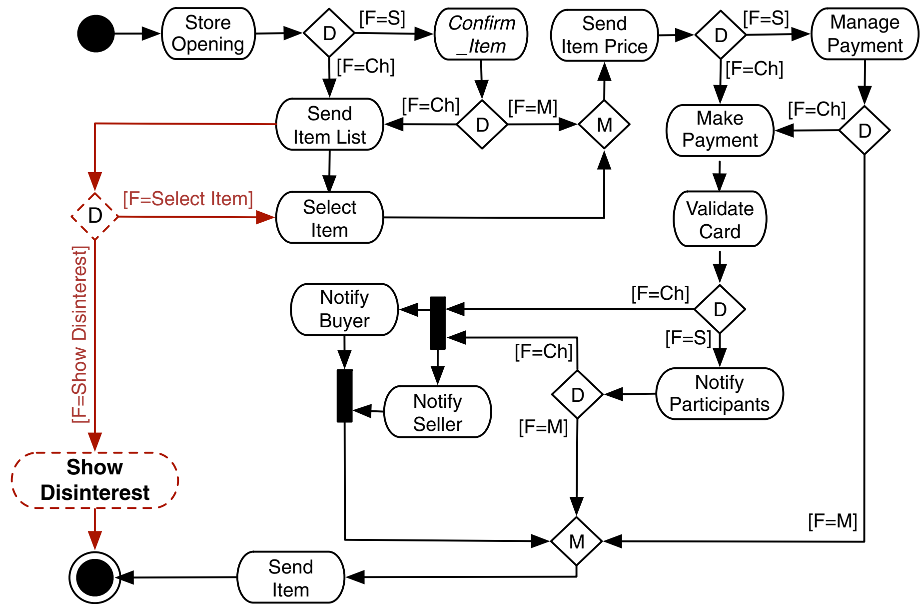


Fig. 12 Activity diagram for the scenario in Fig. 2 with variation

4.3 Maintenance

Scenarios may evolve both throughout and after the analysis phase. For instance, a designer may introduce a variation that includes new goals. For example, the variation to the scenario in Fig. 2 mentioned earlier in sect. 4.1 introduces *“Show_ Disinterest”* as a new goal.

Based on the user study we conducted (Sect. 5.3), incorporating the newly introduced goals by considering the text-based variation requires more mental effort from designers than the activity diagram. The activity diagram provides designers with a context that helps them associating the goals with the existing goal trees.

5 Evaluation

In this section, we outline a controlled experimental evaluation to measure the usefulness that having a complete activity diagram has on a software engineer analysing a Prometheus goal model and scenario. We recruited fifteen participants with varying levels of experience in Prometheus, and gave them several tasks to complete on simple requirements documents, measuring aspects of their performance.

We are primarily interested in evaluating the activity diagrams for understandability—that is, how much the presence of an activity diagram impacts a person's ability to understand the requirements models—and maintainability—that is, how straightforward a design is to maintain given an activity diagram.

5.1 Experimental design

We recruited a total of fifteen participants in the experiment. A pre-evaluation questionnaire,[8] consisting of nine questions, was used to measure their experience with software engineering in general, with agent-oriented software engineering, and the Prometheus methodology. All participants were familiar with intelligent agents (researchers or practitioners in a related area) and most had some experience in requirements analysis; eleven participants had experience in the Prometheus methodology and all but one had experience using some agent-oriented methodology, including Prometheus. All but one of the participants were experienced software developers, with the final participant still holding a degree in computer science.

Each participant was then given several tasks to perform.[9] The tasks asked participants to interpret, verify, and modify a set of requirements models for two simple systems. The presence or absence of the additional activity diagram in the requirements provided was the independent variable. This was the only difference: the remainder of the content in the requirements documents did not change. It is worth noting that the activity diagram provided was automatically generated from the scenario and the goal overview diagram. Then, all the modifications were applied manually on that activity diagram.

To mitigate experience bias, for each participant, exactly one of the requirements documents contained an activity diagram. Thus, each participant was asked to complete three tasks without the aid of an activity diagram for one system, and the same three tasks, as well as an additional fourth task, with an activity diagram for the other system (see below for discussion of the tasks).

To mitigate participant bias, participants were divided into two groups, G1 and G2. G1 received system S1 with an activity diagram, and system S2 without, while G2 received system S1 without an activity diagram, and S2 with an activity diagram.

As Table 2 shows, some participants were more familiar with UML behavioral models (Q2.2)[10] than Prometheus (Q1.3).[11] To mitigate this familiarity bias, all participants were given an optional material to read prior to the experiment (refer to Appendix 2). These two pages short explain the relevant parts of both Prometheus and Activity diagram to the experiments. Also, all the participants were given the opportunity to clarify any point in that optional material prior to the experiment. Note that the aim of this evaluation was to evaluate the benefits of supplementing Prometheus with activity diagram, rather than comparing Prometheus with activity diagram.

Finally, within each group, we balanced out in which order the participants received the systems, to avoid a potential bias in which participants used their experience from the first system to answer questions on the second. That is, half of the participants received the activity diagrams in the first set of tasks, and half received them in the second set.

We measured two dependent variables: time and correctness. For time we simply measured the clock time from start to completion for each task as a proxy for both maintainability and understandability; that is, how much the activity diagram aids software engineers to come up to speed with the semantics of the requirements models, and to modify them. There were no time limits on tasks. For correctness we assessed the participants' answers to each task

[8] Refer to Appendix 1.

[9] Refer to Appendix 2.

[10] Participants were asked to rate their experience with UML behavioural models on a scale of 1–5, with 1 being inexperienced and 5 being expert.

[11] Participants were asked to rate their familiarity with Prometheus on a scale of 1–5, with 1 being very unfamiliar and 5 being very familiar.

Table 2 Summary of the pre-evaluation questionnaire outcomes (refer to Appendix 1)

ID	Q1.1	Q1.2	Q1.3	Q2.1	Q2.2	Q2.3	Q2.4	Q3	Q4	Q5
1	2	5	3	2	2	3	4	11–20	11–20	No
2	4	4	4	4	3	3	4	1–5	1–5	No
3	4	4	3	4	4	4	3	6–10	1–5	No
4	2	5	4	3	2	2	4	11–20	1–5	No
5	4	4	3	3	3	3	3	11–20	1–5	No
6	3	3	2	3	3	3	2	6–10	1–5	Yes
7	4	4	1	4	4	5	4	30+	11–20	Yes
8	3	4	1	4	3	4	4	30+	1–5	No
9	5	4	3	5	5	5	3	11–20	1–5	Yes
10	4	5	5	4	3	3	5	1–5	6–10	No
11	4	4	1	4	4	5	1	11–20	1–5	No
12	5	3	3	5	3	4	2	30+	1–5	No
13	4	4	4	5	5	5	3	11–20	1–5	No
14	3	4	4	4	3	2	4	1–5	1–5	No
15	5	4	3	5	4	5	3	21–29	0	No

to determine whether they had completed the task correctly. This is also used as a proxy for measuring maintainability and understandability; that is, how much the activity diagram impacts the ability to understand and modify the requirements models correctly.

After the tasks had been completed, participants were asked to complete a four question survey[12] asking about their experience, and their perception of the usefulness of activity diagrams. We asked the following four questions, each on a scale of "Strongly Disagree" to "Strongly agree", with a "Neutral" option: Q1: *The activity diagram enables an easy extraction of a possible behaviour path relative to a particular scenario*; Q2: *The time taken to grasp what the entire activity diagram shows is reasonable*; Q3: *It is easy to maintain activity diagrams (easy to incorporate changes including: adding, removing and modifying entities)*; and Q4: *Activity diagrams would be useful in designing the agents of a particular scenario*. Participants were also asked which approach they preferred, and why.

5.2 Tasks

Each participant was asked to perform the following tasks: (1) *verify* a set of five traces, assumed to be extracted from an execution of the system, against the requirements models; (2) *specify* three traces that are valid with respect to the requirements models; (3) *modify* a goal overview diagram based on new requirement proposed by a stakeholder; and (4) *modify* an activity diagram based on new requirement proposed by a stakeholder (only for the system for which an activity diagram was provided). Since the tool developed generates a static diagram, we asked participants to hand-draw the modifications.

The purpose of the first two tasks is to measure understandability of models, while the latter two aim to measure maintainability.

For example, task 1 asked participants to verify whether the following trace from the Sale Transaction system is valid:

[12] Refer to Appendix 3.

Store_Opening, Send_Item_List, Select_Item, Send_Item_Price, Make_Payment, Validate_Card, Notify_Participants, Send_Item.

Task 3 asked participants to modify the goal overview diagram given the new requirement: “After the Buyer agent receives the item list event from the Seller agent, it may select a product or *may show its disinterest*.”

In real terms, these tasks are small, in that each task took only a few minutes to complete. However, while small, they are not trivial—as indicated in the results, a significant proportion of the participants made mistakes.

5.3 Results

In this section, we present and discuss our experiment results.

5.3.1 Task analysis results

Table 3 presents a breakdown of the results for each task. Numbers in each cell for columns labeled 0–5 represent the percentage of participants who scored that number. Scores refer to the number of traces correctly identified. Task 1 had 5 traces, whilst Task 2 had 3 traces; e.g. for task 1 performed without the activity diagram (“✘”), 27 % of the participants scored two out of five. With regards to Task 3 and Task 4, ‘score 0’ means that the task was not achieved, whilst ‘score 1’ means that the task was achieved. The mean and standard deviation for scores and completion time are presented as well. These results show a clear trend that participants scored higher on tasks if they had the help of activity diagrams.

Understandability Recall that tasks 1 and 2 aim to measure the understandability of requirements models with and without activity diagrams. The average score on tasks 1 and 2 are over one point higher when aided by an activity diagram—and there are only a total of three points in task 2. Further to this, with the aid of an activity diagram, almost all of the participants obtained all sub-tasks correct over tasks 1 and 2, compared to only a handful for when not using an activity diagram.

Table 3 Task analysis results

Task	AD	Scores (number as %)						Mean	SD	Mean	SD
		0	1	2	3	4	5	score	score	time	time
T1	✘	0	0	27	7	40	27	3.67	1.18	4:16	2:49
	✔	0	0	0	0	13	87	4.87	0.35	2:39	1:19
T2	✘	27	7	27	40			1.8	1.26	3:04	0:48
	✔	0	7	7	87			2.8	0.56	2:50	1:01
T3	✘	67	33					0.33	0.49	3:08	1:38
	✔	27	73					0.73	0.46	2:18	1:08
T4	✔	0	100					1.0	0.00	1:17	1:04

AD Activity diagram

Further to higher scores in correctness, participants completed task 1 on average around two minutes faster with an activity diagram than without, and with less variation. Completion times for task 2 was slightly faster with the activity.

The reason for these results is clear: with an activity diagram, the specified behaviour of the system can be easily inferred. With just a goal overview diagram and scenario, the interplay between the two models can introduce subtle variations in behaviour that are not obvious. Adding an activity diagram helps to remove much of the ambiguity related to this.

However, the standard deviation in time for task 1 was large due to one participant taking over 6 minutes to complete the task. Further, this participant took the longest out of all participants in all four tasks using the activity diagram, and also took longer in tasks 1–3 with activity diagram than without, as the participant was using both the activity diagram and the scenario with the goal tree. This indicates that the addition of an activity diagram may not always be useful.

Maintainability Recall that tasks 3 and 4 aim to measure the maintainability of requirements models with and without activity diagrams. For task 3, the percentages and average scores show that when aided by an activity diagram, participants were able to correctly modify the goal overview to consider new requirements more often than without the activity diagram. In addition, the average time taken was 50 seconds faster with an activity diagram.

Interestingly, in task 3, one participant did not even commence the task for the system without the activity diagram, despite having done task 3 with the aid of activity diagram already for the other system. The participant commented that they just did not know where to start.

These results add evidence to our hypothesis that activity diagrams are useful aids when maintaining/modifying requirements models. We attribute this mostly to the fact that the participants had a better understanding of the specified behaviour, as also demonstrated by tasks 1 and 2. However, task 3 asks participants to modify the goal overview diagram, so the results provide evidence that the activity diagram is not just useful for characterising specific behaviour traces in the requirements models, but also for considering *sets* of behaviours, which is what they needed to do to update the goal overview diagram.

Task 4 asked participants to modify the activity diagram, so there is no comparison to be made. To assess the correctness of this, we checked whether the updated activity diagram captured the two additional traces, and *only* those two additional traces. Our conclusion for Task 4 was based on the fact that all participants were able to correctly modify the activity diagram as instructed. The results here clearly provide support that maintaining the activity diagrams does not add a large amount of complexity.

Figure 13a shows a box-plot of the scores (out of nine) for all scores. These plots demonstrate that participants did better in the activity-based approach with a median value equals to nine (the maximum score), compared with a median of six in the non-activity approach. Also, the minimum score of the activity-based approach is greater than the median of the score in the non-activity one ($7 > 6$). With respect to the time taken in each approach (Fig. 13b), the activity-based approach takes less time than the non-activity approach. All the minimum, median and maximum values (4:37, 7 and 15:25 respectively) of the time taken in the activity-based approach is less than the non-activity approach (5:06, 10:30 and 22:14 respectively).

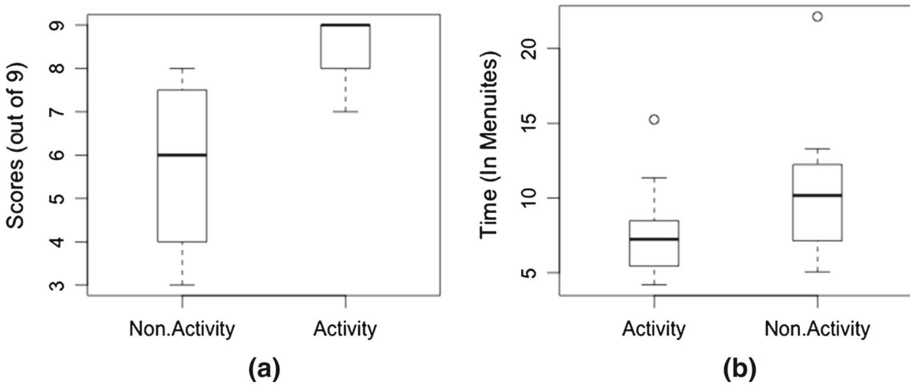


Fig. 13 Box-plots for total scores and total time. **a** Scores (out of nine). **b** Time (minutes)

Table 4 Post-evaluation questionnaire

Qst	SD (%)	D (%)	N (%)	A (%)	SA (%)	Mean	SD
Q1	0	0	7	20	73	4.67	0.62
Q2	0	0	0	53	47	4.47	0.52
Q3	0	0	0	47	53	4.53	0.52
Q4	0	0	0	40	60	4.6	0.51

Responses: [SD] Strongly Disagree = 1, [D] Disagree = 2, [N] Neither = 3, [A] Agree = 4, [SA] Strongly Agree = 5. See Sect. 5.1 for the questions

5.3.2 Post-evaluation questionnaire

Table 4 shows the results for the post-evaluation questionnaire. As stated in Sect. 5.1, we asked participants to rank their experience through answering the following four questions:

(1) The activity diagram enables an easy extraction of a possible behaviour path relative to a particular scenario.
(2) The time taken to grasp what the entire activity diagram shows is reasonable.
(3) It is easy to maintain activity diagrams (easy to incorporate changes including: adding, removing and modifying entities).
(4) Activity diagrams would be useful in designing the agents of a particular scenario.

The results here demonstrate a strong preference for the inclusion of activity diagrams, and that participants felt the diagrams were useful for understanding behaviour, were straightforward to use, and would aid with the design process.

Answers to the open-ended question also further confirm these results:

> "The approach with the activity diagram is preferable. While I prefer (I think) to design agents using the goal overview, analysing possible interactions between agents, sequence of actions etc. is made easier with the activity diagram. Design flow would be easier to find through analysis of activity diagram, as you have a single line to follow rather than multiple goal trees."—Study participant 1.

Some participants noted that it is not the activity diagram itself that is useful, but the activity diagram in combination with the goal overview diagram:

"The one with activity diagram is more helpful, but it is not a substitute for goals. It just makes the trace analysis easier because the sequence of events is integrated regardless of the agent by whom they are caused."—Study participant 7.

Three other participants noted that the usefulness of including the activity diagram is not limited on understanding the behaviour, rather it facilitates the communication between the system analyst and the stakeholders.

"I would prefer to use the approach that uses activity diagram, because it was very useful to understand the behaviour of the system in simple way that could help the requirements analyst to share it with the stakeholders which will help to mitigate the problem of communicating the requirements with clients."—Study participant 12.

5.4 Statistical significance

This sections presents the results of significance tests on our results. Recall that the two measures taken are the (*correctness* and *time*).

Table 5 lists the scores, times, and the means of the scores (out of 9) and time taken for all the participants in the first three tasks of each approach, as well as the confidence intervals at 95 %. Note that we excluded the fourth task from the *activity-based* approach in this comparison, as it does not have its equivalent task in the other approach.

Table 5 Summary of the total results for each participant in both approaches

Participant_ID	Activity diagam		Non-activity	
	Scores (out of 9)	Time (in min)	Scores (out of 9)	Time (in min)
1	9	15:25	8	8:15
2	9	7:33	4	11:35
3	9	8:41	8	22:14
4	8	8:35	4	5:06
5	8	11:36	7	13:30
6	9	11:34	6	13:33
7	8	5:10	3	7:00
8	9	4:37	8	10:17
9	9	5:38	4	12:15
10	7	5:51	8	7:27
11	9	7:21	5	12:33
12	9	7:24	3	6:24
13	8	4:19	7	10:04
14	7	8:54	5	10:17
15	8	6:05	7	6:49
Mean	8.4	7:55	5.8	10:29
CI	0.373	1:41	0.96	2:20
Upper confidence bound	8.77	9:36	6.76	12:49
Lower confidence bound	8.02	6:13	4:84	8:08

CI confidence interval value at a confidence level of 95 %

Given the estimated range of the scores of the activity-based approach ($8.02 \leq mean \leq 8.77$) and the non-activity one ($4.84 \leq mean \leq 6.76$), it is noted that these intervals do not overlap, therefore, we can conclude that the results for the scores are significant at the 95 % level of confidence.

Regarding the time taken in each approach, it is clear that the participants took less time with the activity-based approach, on average, compared to the non-activity approach. However, the confidence intervals for these samples overlap, so further analysis is required.

The sample under analysis is a paired sample, as all the fifteen participants dealt with both approaches. Using a Shapiro test [36], we concluded that the data was not normally distributed. Thus, we used the *Wilcoxon-signed-rank* non-parametric test [23] for our samples. The null hypothesis (H_0) is that there is no difference between the times in the activity-based approach and the non-activity approach, while the alternative (H_1) is that there is a significant difference.

We ran the test in *R* with a significance level of 95 % ($\alpha = 0.05$). The resulting *p*-value is 0.044 (p value ≤ 0.05), so we reject the null hypotheses (H_0). Thus, there is a *significant* difference between the times in both approaches.

5.5 Threats to validity

A main threat to external validity in our experiment is the *scale* of the designs used. The systems used in the study were not large in scale. Larger systems may perhaps result in a more complicated activity diagram that requires more time to understand than the original approach. Thus, further experimentation on larger systems are required to provide more generalisable results.

Since the first two tasks require participants to deal with behavioural runs, *maturation* is an internal threat to the validity of the experiments. The first task asks participants to validate behavioural runs, whilst task 2 requires them to specify possible behavioural runs given the same scenario and goal tree. Thus, participants may use their experience from task 1 in achieving task 2, and hence the correctness and time taken for task 2 may be affected. As a result, the experiments need to distinguish task 1 from task 2, by either varying the goal tree or the scenario, to obtain more reliable results. However, we distinguished these two tasks through rearranging the choices (refer to Appendix 2). We observed all participants, and all of them tackled both tasks independently (i.e. they did not rely on their experience from task 1 in achieving task 2). In fact, all the participants specified different behavioural runs in task 2 than the ones mentioned in task 1.

6 Related work

In the context of AOSE, there has been a strong focus on models for requirements, but little research into the relationships between these models. Considering the six AOSE methodologies mentioned in Sect. 2, they all support the requirements gathering and analysis process. Further, most of the methodologies share similar requirements specification elements [12]. Such processes results in a variety of artefacts, and hence the information is scattered across these artefacts. Despite the fact that these methodologies do offer development environments through their supported tools [2,17,19,29,34], they do not offer the ability to merge this information from the different artefacts into one cohesive structure. Further, some of them, such as Prometheus and ROADMAP do not provide a structured approach for specifying variations to the requirements, whilst other, such as MaSE, do.

In [2] a comparison between three methodologies (ROADMAP, Prometheus, and MaSE) is conducted. The comparison shows that all these methodologies share the notions of goal hierarchy and use cases in specifying requirements. However, they provide little, if any, support for structured specification of variations, and merging different artefacts into a single representation. As shown in our evaluation, this single representation enhances the understandability and maintainability of the requirements.

Even though MaSE uses sequence diagrams to represent the textual use case scenarios. In other words, it transforms the textual use case into sequence diagram. Even though the methodology transforms the textual description into a diagrammatic representation, it does not consider the goal hierarchy in this transformation process [15]. Also, the tool support (agent tool [13]) does not automatically generate the sequence diagram equivalent to a given scenario which can be error-prone. Also, the generated sequence diagram will lack the variations that are resulted by the relevant information to the scenario from the goal tree.

There is a single attempt, extending Prometheus, to enhance the approach of specifying requirements. In [39] the authors proposed a more structured way for specifying requirements in Prometheus. Their focus was on generating scenario-based test cases for run time testing of agent systems. They provided a means for specifying variations in terms of actions and percepts and also showed how scenarios may be traced to the detailed design of the agents for coverage. Future work, would involve investigating a similar traceability approach to the activity diagram based approach presented in this work.

In ROADMAP/AOR [38], scenarios already support the structures considered in our work, including sequence, branching and parallelism. The notation used is tabular rather than graphical, but can be used to specify traces in a similar spirit to activity diagrams. However, these scenarios are not linked directly to goal models, and hence the methodology does not merge them.

With respect to Gaia, there is an attempt to extend the methodology through incorporating the AUML modelling notations including sequence diagram and activity diagrams [16]. The authors presented such integration through the development of a flexible manufacturing control system following the Gaia methodology. As stated in Sect. 2, in Gaia the roles (functions) of the intended system are specified as role models as an initial step. Then, the interaction models are used to model the interactions between agents, and the dependency between the different roles. In [16], the AUML sequence diagrams are used to model the interactions. However, these diagrams are manually generated. Also, the authors used activity diagrams to manually model the internals of each agent based on its functions (*roles*).

Alam et al. in [3] propose guidelines in which to transform the early requirements models in the secure Tropos into UMLsec[13] use-case diagram. Such transformation is meant to show the dependency between the actors and the functionalities that require security in the intended system. Thus, they provided a mechanism to elicit information from a bigger model (early requirements model) into one cohesive model (use-case diagram). They used the Kent Modelling Transformation Language (KMTL) [1] in the transformation, however, the authors did not indicate whether this transformation was automatic or not. Also, their proposed approach does not allow designers to edit such models, and incorporate variations.

There is other relevant work on automating the transformation of use-cases into UML activity diagrams. Yue et al. in [48] propose a sophisticated algorithm to automatically transform use cases into UML activity diagrams. The use-cases supported in their approach must follow a tabular format that includes the basic flow and the alternative ones. They adopt different natural language processing techniques in analysing the free-text description of

[13] UMLsec is an extension to UML notation to enable the development of secure systems [27].

the use-cases. Thus, the use-cases description must follow the RUCM template that has 26 well-defined restriction rules [47]. However, the format of scenarios in our approach does not restrict designers in specifying the steps. Also, the proposal in [48] does not consider other artefacts than the use-cases, whilst our approach considers the goal hierarchy.

In [21], the authors developed an approach to automate the process of generating an activity diagram out of a use-case scenario. They used the QVT-rational language (*Query/View/Transformation*) in the model transformation. The authors provide designers with a metamodel that must be followed in specifying use-case scenarios. This metamodel allows designers to specify the core elements of the intended scenario including: participants, pre-conditions, post-conditions, the main flow of steps and any alternatives. Even though the metamodel provided in their work is similar to the structure of scenarios in Prometheus, it does not allow the specification of parallel steps. However, in Prometheus, scenarios can implicitly capture parallelism through having a goal step in the scenario that has children with the *undirected-conjunctive* decomposition. Also, and unlike our approach, the metamodel does not allow designers to merge any relevant information to the specified use-case scenario from other artefacts.

7 Conclusion

In this paper, we have proposed a new approach for specifying requirements in the Prometheus methodology. The approach is to introduce activity diagram to be used as a coherent structure that merges the steps of a given scenario along with their related information from the goal overview diagram. Although our approach is grounded in Prometheus, it is applicable to any methodology that link the notions of goal hierarchies and use case scenarios in the system specification phase.

We discussed the benefits our approach offers designers with respect to different design aspects. First, our approach enables designers to specify more control fragments such as parallel and variations in a more structured manner. Second, it can be used to maintain consistency between different artifacts.

Our evaluation showed that including activity diagrams leads to better understanding of the specification by giving a more holistic view on what the intended system is meant to achieve and how it should behave, and provides assistance when performing maintenance on the system. Participants in our experiment unanimously agreed that the inclusion of the activity diagram improved their ability to understand the requirements models. Given this, we recommend the Prometheus methodology, and indeed the other AOSE methodologies, to include activity diagrams as an integrated feature in the methodology.

As an extension to this work, we are investigating ways to provide *round-trip engineering* in our approach through designing and implementing an algorithm to automate the process for propagating information into the goal hierarchy from changes in the activity diagrams. We also plan to investigate how the activity diagrams can be used for automated verification of design models and implementations. Since scenarios may include alternatives and loops, we are investigating the inclusion of swim-lanes in the approach to improve the readability of the generated activity diagram. At this stage, our approach considers one scenario at a time, as each scenario describes a basic run in the intended system, and is not linked to other scenarios. In fact, Prometheus does not offer a way to associate different scenarios. The inclusion of multiple scenarios in one activity diagram represents another extension to our approach. According to the results of the evaluation conducted, some participants totally

relied on the activity diagram, whilst other participants used both activity diagram and goal overview diagram. Thus, a potential item of further work would be to reconsider some aspects of Prometheus.

Acknowledgments Y. Abushark acknowledges King Abdulaziz University for scholarship. J. Thangarajah acknowledges the support of the Australian Research Council under Discovery Grant DP1094627.

Appendix 1: Pre-evaluation questionnaire

The main purpose of the pre-questionnaire is to assess the user's experience in the filed of AOSE, specifically the Prometheus methodology. Also, it aims to assess your experience in the UML activity diagrams.

On a scale of 1–5 (with 1 being Very Unfamiliar, and 5 being Very Familiar) Please rate your familiarity with:

(1a) Software Requirements Analysis Concepts: Very Unfamiliar **1 2 3 4 5** Very Familiar
(1b) Intelligent Agents: Very Unfamiliar **1 2 3 4 5** Very Familiar
(1c) The Prometheus AOSE Methodology: Very Unfamiliar **1 2 3 4 5** Very Familiar

On a scale of 1–5 (with 1 being inexperienced, and 5 being expert) Please rate your familiarity with:

(2a) Software Analysis and Design : inexperienced **1 2 3 4 5** expert
(2b) UML Behavioural Models: inexperienced **1 2 3 4 5** expert
(2c) Use Case Scenarios: inexperienced **1 2 3 4 5** expert
(2c) BDI-Agent System Modelling: inexperienced **1 2 3 4 5** expert
(3) How many software systems have you designed (not including agent-oriented)?

☐ 0
☐ 1–5
☐ 6–10
☐ 11–20
☐ 21–29
☐ 30+

(4) How many agent software systems have you designed?

☐ 0
☐ 1–5
☐ 6-10
☐ 11–20
☐ 21–29
☐ 30+

(5) Do you have any expertise in other AOSE methodologies other than Prometheus? List them if yes

☐ No
☐ Yes ____________________________________

Appendix 2: Experiment sheet

Experiment prerequisite materials *(Optional)*

– Specifying requirements in the Prometheus methodology

In Prometheus, requirements are specified using goals and use case scenarios. A scenario is a sequence of steps that describe a particular run of the system. These steps are of different types ranging from sub-scenarios to the goals that need to be achieved by the system. Although a scenario is a single sequence of steps, the result is a set of sequences, because a goal step can be realised by implementing its children (more generally, its descendants), or its parent from the goal model of the system (goal overview diagram). For example the scenario in Fig. 14a, the "Invite_Reviewers" goal step could be realised either by the goal step itself or along with "Invite_Reviewers_Via_Email"; or "Invite_Reviewers_Via_Portal" goal steps. Also, a scenario may have some variations at some point. Such variations are specified in a free-text manner. consequently, these variations may introduce new goals and/or modifying the compositions of existing goals in the goal overview diagram.

– Review scenario of the conference management system

The conference management system is an agent-based system that helps in managing the different phases of the international conferences including: submission, review, decision and paper collection.In the submission phase, the system should be able to assign a number for each submission and provide receipts to authors. After the specified submission deadline, the system assigns papers to the reviewers. After receiving the reviews, a decision should be made, by the system, about accepting or rejecting the papers with notifying authors. Then, the system collects the accepted papers and prints them in a form of a conference proceeding. As Fig. 14 shows, the review scenario (Fig. 14a) includes number of steps ranging from percepts the system perceives to goals to pursue. Also, one text-based variation is specified after the second step. Figure 23b outlines the goals and sub-goals required to successfully review the papers. As can be seen in the figure, there are three types of goal decomposition: OR, undirected-AND (denoted by AND) and directed-AND (denoted by AND with dashed arrows between the child goals indicating the ordering constraints, e.g. Invite Reviewers to Collect Prefs).

– UML activity diagrams

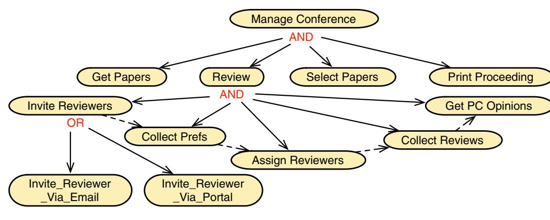
UML activity diagrams are typically used for business process modelling, for modelling the logic captured by a single use case scenario. Activity diagrams provide designers with a range

	Type	Name	Role
1	Percept	Review_Phase	Review Management
2	Goal	Invite Reviewers	Review Management
3	Action	Send_Invitations	Review Management
4	Percept	Reviewers_Preferences	Review Management
5	Goal	Collect Prefs	Assignment
6	Goal	Assign Reviewers	Assignment
7	Action	Give_Assignments	Assignment
8	Percept	Review_Report	Review Management
9	Goal	Collect Reviews	Review Management
10	Goal	Get PC Opinions	Review Management

Variations

At step 2, if the researches did not have access to the portal, the system should be able invite them through email

(a)



(b)

Fig. 14 Review scenario and system's goal overview diagram. **a** Scenario description. **b** Goal overview diagram

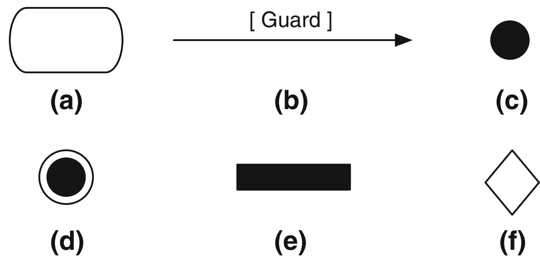


Fig. 15 Adopted UML activity diagram notations

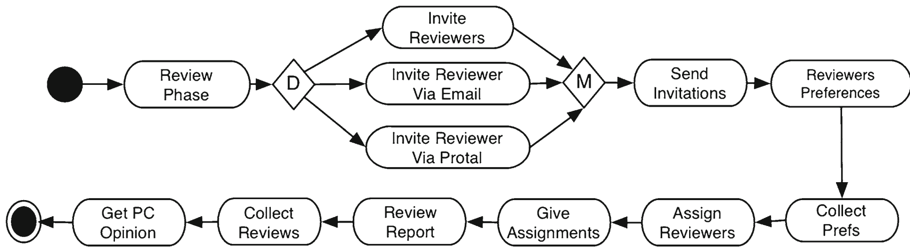


Fig. 16 Equivalent AD for the scenario in 14a

of graphical notations for modelling activity diagrams including: activity nodes and activity edges. The following six notations (Fig. 15) are necessary for this project:

(1) *Action Node* it is the fundamental and the executable node in the activity diagrams. It is notated by a rectangular shape with rounded edges.
(2) *Regular Activity Edge* It is a directed connection that shows the flow between the different actions within an activity diagram.
(3) *Initial Node* It is a control node that initiates the execution of an activity.
(4) *Final Node* It is a control node that shows the end of an activity.
(5) *Fork/Join* It is control node that splits the execution flow into multiple threads to be executed independently, and then synchronises them.
(6) *Decision/Merge* It is control node that splits the execution of an activity into multiple alternate flows with only one flow to be executed.

Revisiting the conference management system example, Fig. 16 shows the equivalent activity diagram for the scenario along with its variation. In fact, Activity Diagram is a coherent structure that merges the scenario steps along with their related information from the goal overview diagram.

System1: Trading Agent System

The trading agent system is an agent-based system that captures the process that takes place in a sale transaction. The system has three agents including: seller, buyer and banker. The seller agent must send the list of products to the buyer agent when it receives the "store_opening" percept. The buyer agent then selects a product. After that, the seller agent should send the buyer the price of the selected item.

Then, the buyer agent should proceed with the payment through the banker agent. The banker agent then processes the payment and notifies both the seller and the buyer about the payment process outcomes (approved or denied). The order of these notifications is

not important (e.g. seller first and buyer second or the other way around). In the case of an approved payment, the seller must send the item to the buyer.

NOTE The provided designs across the tasks are possible designs and do not necessarily represent good ones.

NOTE Activity Diagram is a coherent structure that merges the scenario steps along with their related information from the goal overview diagram.

Remember A goal step can be realised through its children, or in some cases through its parent

Task 1

Assuming that the scenario description (Fig. 17) along with the goal overview (Fig. 18) capture the specifications of the system, determine the valid and invalid traces.
Remember: Activity diagram acts as a coherent structure that merges the steps of the scenario along with its related information from the goal overview diagram, and hence it can be used in achieving the task (Fig. 19).

Fig. 17 Sale transaction scenario description

	Type	Name
1	Percept	Store_Opening
2	Goal	Send_Item_List
3	Goal	Select_Item
4	Goal	Send_Item_Price
5	Goal	Make_Payment
6	Goal	Manage_Payment
7	Goal	Send_Item

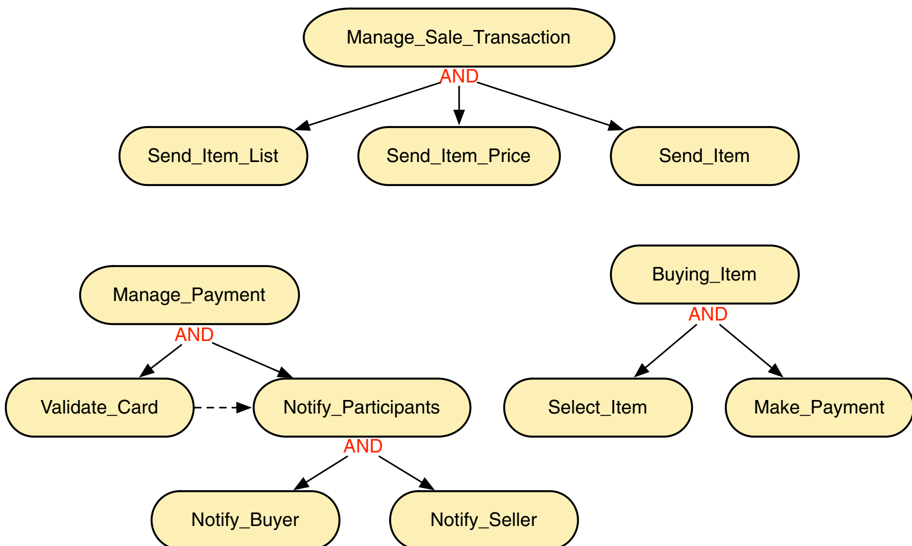


Fig. 18 System goal overview diagram

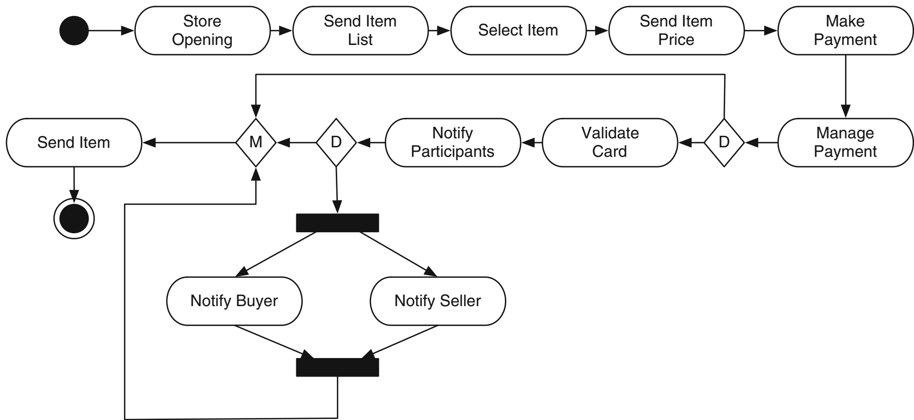


Fig. 19 Activity diagram

	Execution trace 1	Execution trace 2	Execution trace 3	Execution trace 4	Execution trace 5
Token 1	Store_Opening	Store_Opening	Store_Opening	Store_Opening	Store_Opening
Token 2	Send_Item_List	Send_Item_List	Send_Item_List	Send_Item_List	Send_Item_List
Token 3	Select_Item	Buying_Item	Select_Item	Select_Item	Select_Item
Token 4	Send_Item_Price	Send_Item_Price	Send_Item_Price	Send_Item_Price	Send_Item_Price
Token 5	Make_Payment	Make_Payment	Make_Payment	Make_Payment	Make_Payment
Token 6	Validate_Card	Manage_Payment	Manage_Payment	Manage_Payment	Manage_Payment
Token 7	Notify_Participants	Validate_Card	Validate_Card	Validate_Card	Validate_Card
Token 8	Send_Item	Notify_Participants	Notify_Participants	Notify_Participants	Notify_Participants
Token 9	-	Send_Item	Send_Item	Notify_Seller	Notify_Buyer
Token 10	-	-	-	Notify_Buyer	Notify_Seller
Token 11	-	-	-	Send_Item	Send_Item
Answer	☐ Valid ☐ Invalid	☐ Valid ☐ Invalid	☐ Valid ☐ Invalid	☐ Valid ☐ Invalid	☐ Valid ☐ Invalid

Task 2

According to the basic run (scenario in Fig. 20), "Manage_Payment", "Send_Item" is one of the possible paths after posting "Make_Payment" goal. Specify, if possible, another three paths by numbering the tokens based on their order (refer to the example column) (Figs. 21, 22).

Fig. 20 Sale transaction scenario description

	Type	Name
1	Percept	Store_Opening
2	Goal	Send_Item_List
3	Goal	Select_Item
4	Goal	Send_Item_Price
5	Goal	Make_Payment
6	Goal	Manage_Payment
7	Goal	Send_Item

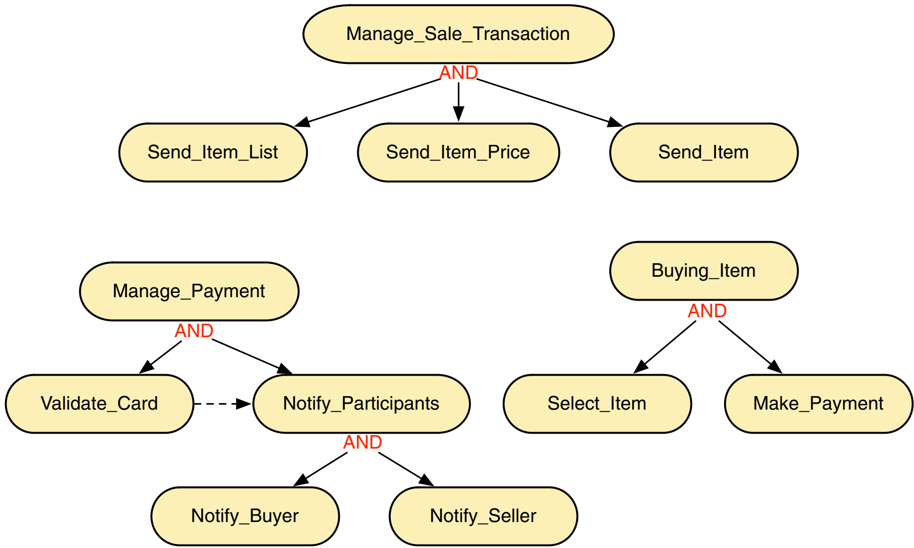


Fig. 21 System goal overview diagram

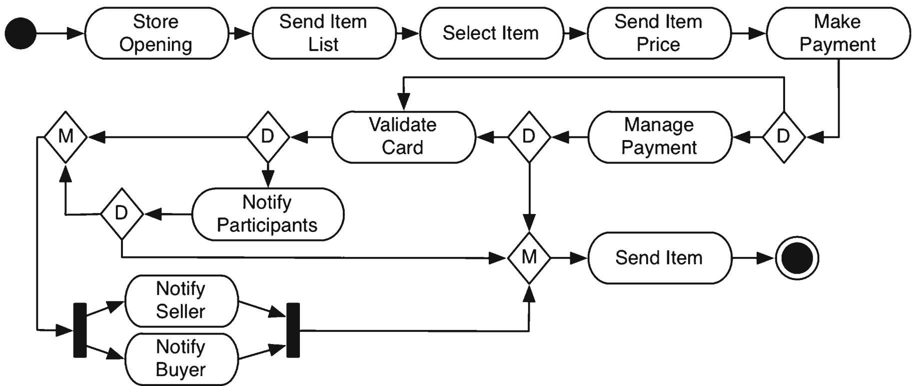


Fig. 22 Activity diagram

Example (this path is based on the basic run)	Path 1 ☐ Not Possible	Path 2 ☐ Not Possible	Path 3 ☐ Not Possible
(___)Manage Sale Transaction (___)Validate Card (___)Notify Seller (___)Notify Participants (___)Notify Buyer (___)Buying Item (___)Send Item List (___)Select Item (___)Send Item Price (___)Make Payment (**1**)Manage Payment (**2**)Send Item (___)Store Opening	(___)Manage Sale Transaction (___)Validate Card (___)Notify Seller (___)Notify Participants (___)Notify Buyer (___)Buying Item (___)Send Item List (___)Select Item (___)Send Item Price (___)Make Payment (___)Manage Payment (___)Send Item (___)Store Opening	(___)Manage Sale Transaction (___)Validate Card (___)Notify Seller (___)Notify Participants (___)Notify Buyer (___)Buying Item (___)Send Item List (___)Select Item (___)Send Item Price (___)Make Payment (___)Manage Payment (___)Send Item (___)Store Opening	(___)Manage Sale Transaction (___)Validate Card (___)Notify Seller (___)Notify Participants (___)Notify Buyer (___)Buying Item (___)Send Item List (___)Select Item (___)Send Item Price (___)Make Payment (___)Manage Payment (___)Send Item (___)Store Opening

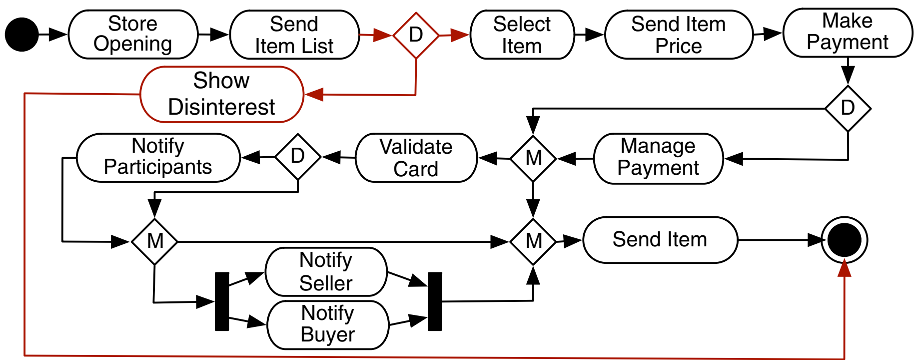
Task 3

One of the stakeholders wants the buyer agent to be able to show its disinterest in a product. Thus, a variation to the scenario is appeared as follows (Fig. 23).

Your task is to add the new goal/s introduced by the variation in the goal overview diagram (Fig. 24).

	Type	Name
1	Percept	Store_Opening
2	Goal	Send_Item_List
3	Goal	Select_Item
4	Goal	Send_Item_Price
5	Goal	Make_Payment
6	Goal	Manage_Payment
7	Goal	Send_Item

(a)



(b)

Fig. 23 Scenario and activity siagram. **a** Scenario. **b** Activity diagram

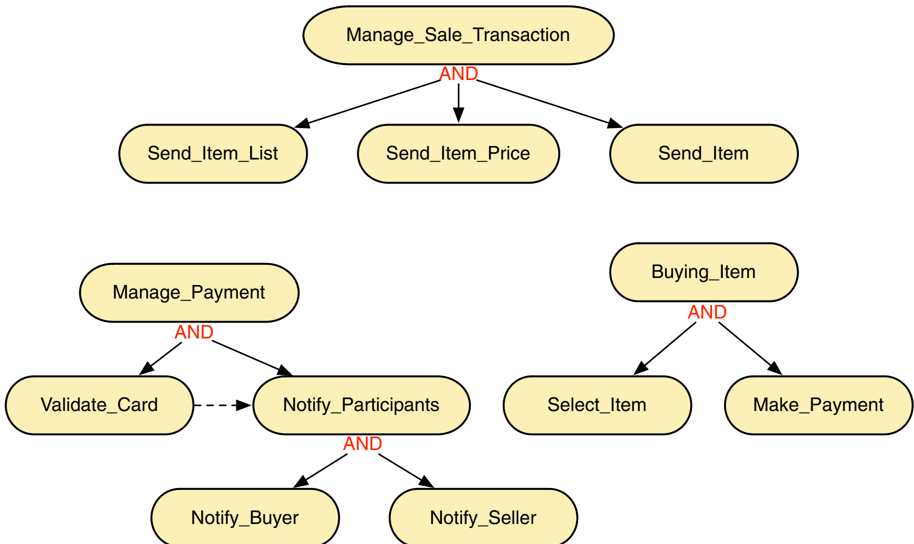


Fig. 24 System goal overview diagram

Task 4

Considering the two traces below, the appearance of Trace 2 is due to a change requested by the stakeholders. Your task is to annotate the activity diagram in Fig. 27 to capture both traces in the table below (Figs. 25, 26).

Token#	Trace 1	Trace 2
1	Store_Opening	Store_Opening
2	Send_Item_List	Send_Item_List
3	Select_Item	Select_Item
4	Send_Item_Price	Send_Item_Price
5	Make_Payment	Make_Payment
6	Manage_Payment	Manage_Payment
7	Send_Item	Cancel_Order

Fig. 25 Scenario description

	Type	Name
1	Percept	Store_Opening
2	Goal	Send_Item_List
3	Goal	Select_Item
4	Goal	Send_Item_Price
5	Goal	Make_Payment
6	Goal	Manage_Payment
7	Goal	Send_Item

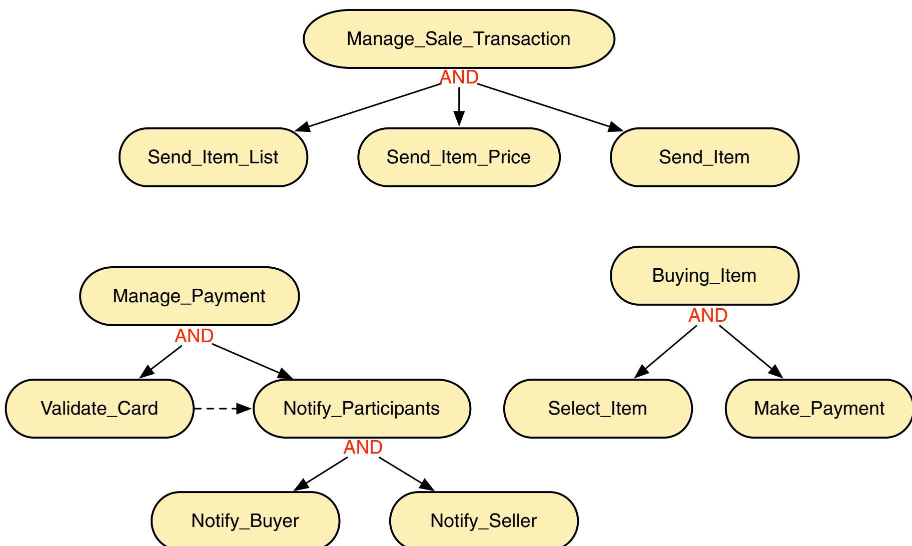


Fig. 26 System's goal overview

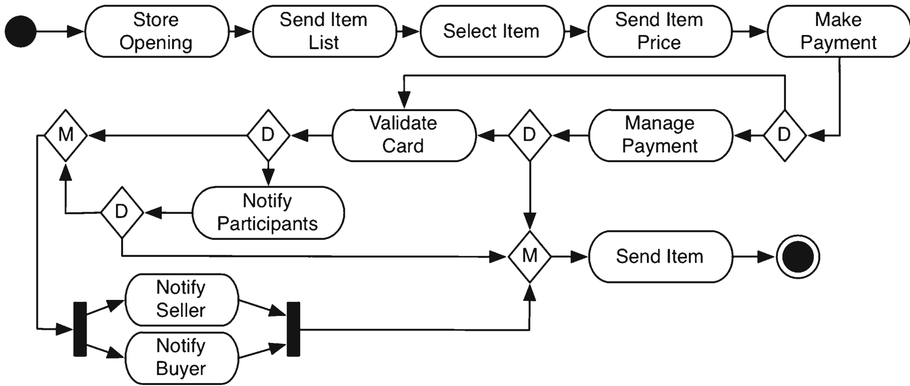


Fig. 27 Activity diagram

System2: Auction System

The auction system is a small agent-based system that simulates the activities that take place in an auction. The system has five agents including: auctioneer, three bidders and banker. The Auction has seven items to bid upon (one item at each round).

The Auctioneer agent should announce the start of the action to the bidders after it receives the "new_auction percept from the environment. The percept includes information about the item to bid and its reserve value.

Then, the bidders should calculate and place their bids. After that, the auctioneer agent should decide on the winning bid and notify the bidders about its decision (who won). Bidders should then update their beliefs with such decision (either winning or lost).

NOTE The provided designs across the tasks are possible designs and not necessarily represent good ones.

Remember A goal step can be realised through its children, or in some cases through its parent

Task 1

Assuming that the scenario description below along with the goal overview (Fig. 29) capture the specifications of the system, determine the valid and invalid traces traces (Fig. 28).

Fig. 28 Auction scenario description

	Type	Name
1	Percept	Start_Auction
2	Goal	Announce_New_Auction
3	Goal	Calculate_Bid
4	Goal	Place_Bid
5	Goal	Identify_Winner
6	Goal	Announce_Winner
7	Goal	Log_Decision

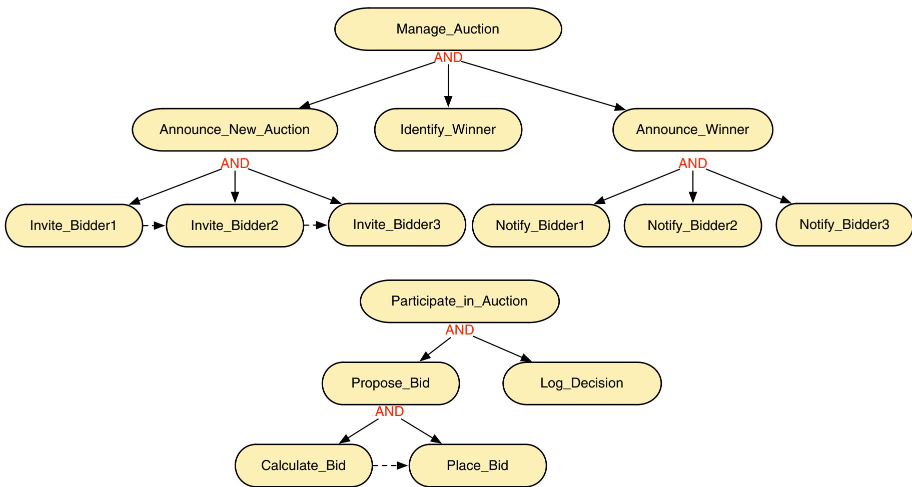


Fig. 29 System goal overview diagram

	Execution trace 1	Execution trace 2	Execution trace 3	Execution trace 4	Execution trace 5
Token 1	Start_Auction	Start_Auction	Start_Auction	Start_Auction	Start_Auction
Token 2	Announce_New_ Auction	Announce_New_ Auction	Announce_New_ Auction	Announce_New_ Auction	Announce_New_ Auction
Token 3	Calculate_Bid	Invite_Bidder1	Propose_Bid	Calculate_Bid	Invite_Bidder1
Token 4	Place_Bid	Invite_Bidder2	Calculate_Bid	Place_Bid	Invite_Bidder3
Token 5	Identify_winner	Invite_Bidder3	Place_Bid	Identify_Winner	Invite_Bidder2
Token 6	Announce_Winner	Calculate_Bid	Identify_Winner	Announce_Winner	Propose_Bid
Token 7	Log_Decision	Place_Bid	Announce_Winner	Notify_Bidder1	Calculate_Bid
Token 8	-	Identify_Winner	Notify_Bidder1	Notify_Bidder3	Place_Bid
Token 9	-	Announce_Winner	Notify_Bidder3	Notify_Bidder2	Identify_Winner
Token 10	-	Log_Decision	Notify_Bidder2	Log_Decision	Announce_Winne
Token 11	-	-	Log_Decision	-	Log_Decision
Answer	☐ Valid ☐ Invalid	☐ Valid ☐ Invalid	☐ Valid ☐ Invalid	☐ Valid ☐ Invalid	☐ Valid ☐ Invalid

Task 2

According to the basic run (scenario in Fig. 30), “Announce_New_Auction”, “Propose_Bid”, “Identify_Winner”, “Announce_Winner” and “Log_Decision” is one of the possible paths after receiving “Start_Auction” percept. Specify, if possible, another three paths by numbering the tokens based on their order (refer to the example column) (Fig. 31).

Fig. 30 Auction scenario description

	Type	Name
1	Percept	Start_Auction
2	Goal	Announce_New_Auction
3	Goal	Calculate_Bid
4	Goal	Place_Bid
5	Goal	Identify_Winner
6	Goal	Announce_Winner
7	Goal	Log_Decision

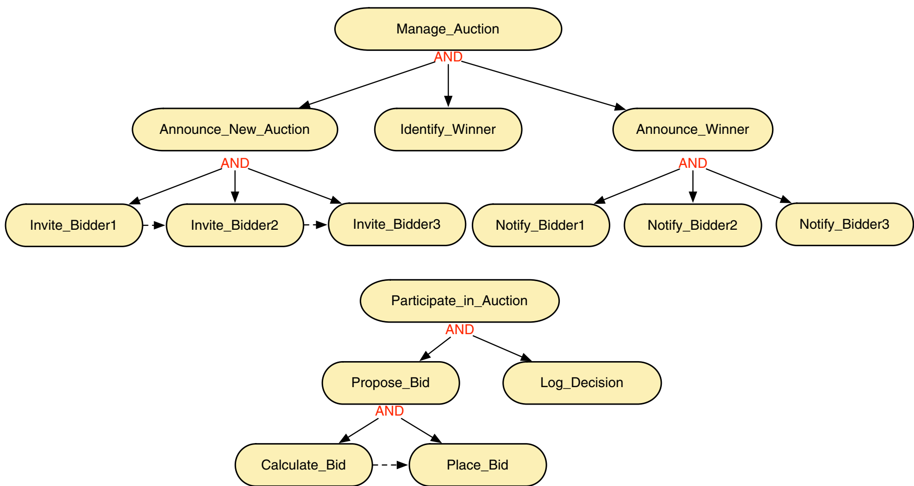


Fig. 31 System goal overview diagram

Example (this path is based on the basic run)	Path 1 ☐ Not Possible	Path 2 ☐ Not Possible	Path 3 ☐ Not Possible
(___)Manage_Auction	(___)Manage_Auction	(___)Manage_Auction	(___)Manage_Auction
(**1**)Announce_New_Auction	(___)Announce_New_Auction	(___)Announce_New_Auction	(___)Announce_New_Auction
(___)Invite_Bidder1	(___)Invite_Bidder1	(___)Invite_Bidder1	(___)Invite_Bidder1
(___)Invite_Bidder2	(___)Invite_Bidder2	(___)Invite_Bidder2	(___)Invite_Bidder2
(___)Invite_Bidder3	(___)Invite_Bidder3	(___)Invite_Bidder3	(___)Invite_Bidder3
(**3**)Identify_Winner	(___)Identify_Winner	(___)Identify_Winner	(___)Identify_Winner
(**4**)Announce_Winner	(___)Announce_Winner	(___)Announce_Winner	(___)Announce_Winner
(___)Notify_Bidder1	(___)Notify_Bidder1	(___)Notify_Bidder1	(___)Notify_Bidder1
(___)Notify_Bidder2	(___)Notify_Bidder2	(___)Notify_Bidder2	(___)Notify_Bidder2
(___)Notify_Bidder3	(___)Notify_Bidder3	(___)Notify_Bidder3	(___)Notify_Bidder3
(___)Participate_in_Auction	(___)Participate_in_Auction	(___)Participate_in_Auction	(___)Participate_in_Auction
(**2**) Propose_Bid	(___)Propose_Bid	(___)Propose_Bid	(___)Propose_Bid
(___)Calculate_Bid	(___)Calculate_Bid	(___)Calculate_Bid	(___)Calculate_Bid
(___)Place_Bid	(___)Place_Bid	(___)Place_Bid	(___)Place_Bid
(**5**) Log_Decision	(___) Log_Decision	(___) Log_Decision	(___) Log_Decision
(___)Start_Auction	(___)Start_Auction	(___)Start_Auction	(___)Start_Auction

Task 3

One of the stakeholders wants the bidder agents to be able to request a loan from the banker agent in the case of three successive losses (Fig. 33). Thus, a variation to the scenario in Fig. 32 is appeared as follows:

> In the case when a bidder agent successively loses the auction three times, it needs to **apply for bank loan**, rather than proposing a new bid.

Your task is to add the new goal/s introduced by the variation above in the goal overview diagram.

Fig. 32 Auction scenario description

	Type	Name
1	Percept	Start_Auction
2	Goal	Announce_New_Auction
3	Goal	Propose_Bid
4	Goal	Identify_Winner
5	Goal	Announce_Winner
6	Goal	Log_Decision

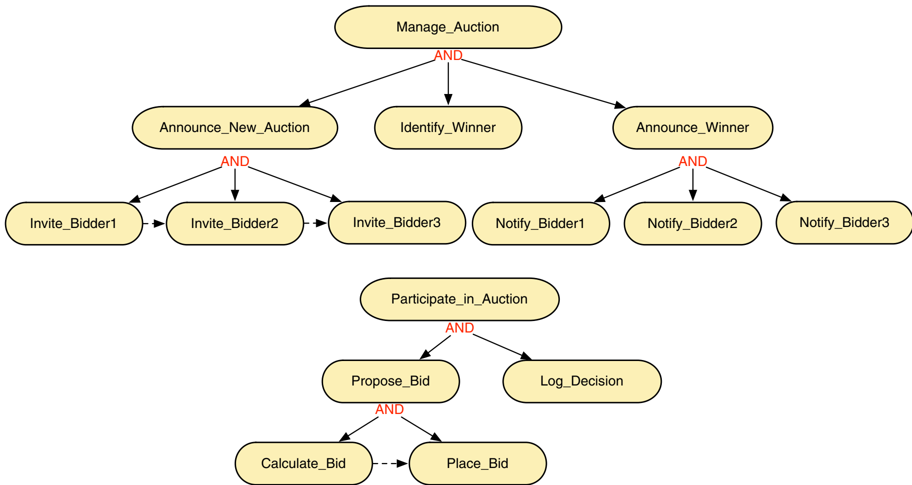


Fig. 33 System goal overview diagram

Appendix 3: Post-evaluation questionnaire

This post test questionnaire is designed to obtain the user's feedback on their experience in both approaches.

Using the following rating sheet, please select the number closest to the term that most closely matches your feeling about the activity diagram as an extra artefact along with scenario and goal overview diagram.

(1) **Enables an easy extraction of a possible behaviour path relative to a particular scenario**

[ ]Strongly Agree [ ]Agree [ ] Neutral [ ]Disagree [ ]Strongly Disagree

(2) **The time taken to grasp what the entire activity diagram shows is reasonable**

[ ]Strongly Agree [ ]Agree [ ] Neutral [ ]Disagree [ ]Strongly Disagree

(3) **It is easy to maintain activity diagrams (easy to incorporate changes including: adding, removing and modifying entities)**

[ ]Strongly Agree [ ]Agree [ ] Neutral [ ]Disagree [ ]Strongly Disagree

(4) **Activity Diagram is useful in designing the agents of a particular scenario.**

[ ]Strongly Agree [ ]Agree [ ] Neutral [ ]Disagree [ ]Strongly Disagree

In the experiment you followed two approaches in specifying requirements: the approach without activity diagram and the one with activity diagram. Which Approach would you prefer to use? and why.

References

1. Akehurst, D. H., Howells, W. G., & McDonald-Maier, K. D. (2005). Kent model transformation language. In: *Proceedings of model transformations in practice workshop (MTIP) at MoDELS conference*, Montego Bay.
2. Al-Hashel, E., Balachandran, B. M., & Sharma, D. (2007). A comparison of three agent-oriented software development methodologies: ROADMAP, Prometheus, and MaSE. *Knowledge-Based Intelligent Information and Engineering Systems* (pp. 909–916). Berlin: Springer.

3. Alam, M. N., Hossain, S., & Alam, K. N. E. (2015). Use case application in requirements analysis using secure tropos to umlsec-security issues. *International Journal of Computer Applications*, *109*(4), 21–25.
4. Beydoun, G., Low, G., Henderson-Sellers, B., Mouratidis, H., Gomez-Sanz, J. J., Pavó, J., et al. (2009). FAML: a generic metamodel for MAS development. *IEEE Transactions on Software Engineering*, *35*(6), 841–863.
5. Bolloju, N., & Sun, S. X. (2012). Benefits of supplementing use case narratives with activity diagramsan exploratory study. *Journal of Systems and Software*, *85*(9), 2182–2191.
6. Bratthall, L., & Wohlin, C. (2002). Is it possible to decorate graphical software design and architecture models with qualitative information?-an experiment. *IEEE Transactions on Software Engineering*, *28*(12), 1181–1193.
7. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., & Mylopoulos, J. (2004). Tropos: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-agent Systems JAAMAS*, *8*(3), 203–236.
8. Cossentino, M. (2005). From requirements to code with the PASSI methodology. *Agent-Oriented Methodologies*, *3690*, 79–106.
9. Dam, H. K., & Winikoff, M. (2013). Towards a next-generation AOSE methodology. *Science of Computer Programming*, *78*(6), 684–694.
10. Dardenne, A., Van Lamsweerde, A., & Fickas, S. (1993). Goal-directed requirements acquisition. *Science of Computer Programming*, *20*(1), 3–50.
11. Deloach, S. (2004). The MaSE methodology. *Methodologies and Software Engineering for Agent Systems*, *11*, 107–125.
12. DeLoach, S., Padgham, L., Perini, A., & Susi, A. (2009). Using three aose toolkits to develop a sample design. *International Journal of Agent-Oriented Software Engineering IJAOSE*, *3*(4), 416–476.
13. DeLoach, S. A. (2001). Analysis and design using MaSE and agentTool. Technical report, DTIC Document.
14. DeLoach, S. A., & Garcia-Ojeda, J. C. (2010). O-MaSE: A customisable approach to designing and building complex, adaptive multi-agent systems. *International Journal of Agent-Oriented Software Engineering IJAOSE*, *4*(3), 244–280.
15. DeLoach, S. A., Wood, M. F., & Sparkman, C. H. (2001). Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, *11*(3), 231–258.
16. Duran-Faundez, C., Ramos, M., & Rodriguez, P. (2015). Applying gaia and auml for the development of multiagent-based control software for flexible manufacturing systems: addressing methodological and implementation issues. *Software: Practice and Experience*, *45*, 1719–1737.
17. Garcia-Ojeda, J. C., DeLoach, S. A., et al. (2009). agentTool III: From process definition to code generation. In: *Proceedings of the 8th international conference on autonomous agents and multiagent systems* (vol. 2, pp. 1393–1394). International Foundation for Autonomous Agents and Multiagent Systems.
18. Gomez-Sanz, J. J., & Fuentes-Fernández, R. (2015). Understanding agent-oriented software engineering methodologies. *The Knowledge Engineering Review*, *30*(04), 375–393.
19. Gomez-Sanz, J. J., Fuentes, R., Pavón, J., & García-Magariño, I. (2008). INGENIAS development kit: A visual multi-agent system development environment. In: *Proceedings of the 7th international joint conference on autonomous agents and multiagent systems: Demo papers* (pp. 1675–1676). International Foundation for Autonomous Agents and Multiagent Systems.
20. Gross, A., & Doerr, J. (2009). Epc vs. uml activity diagram-two experiments examining their usefulness for requirements engineering. In: *17th IEEE international requirements engineering conference, 2009. RE'09* (pp. 47–56). Atlanta: IEEE.
21. Gutiérrez, J. J., Nebut, C., Escalona, M. J., Mejías, M., & Ramos, I. M. (2008). Visualization of use cases through automatically generated activity diagrams. *Model driven engineering languages and systems* (pp. 83–96). Berlin: Springer.
22. Henderson-Sellers, B. & Giorgini, P. (Eds). (2005). *Agent-oriented methodologies*. Hershey: Idea Group Publishing.
23. Hollander, M., Wolfe, D. A., & Chicken, E. (2013). *Nonparametric statistical methods*. New York: Wiley.
24. Jacobson, I. (1992). *Object-oriented software engineering: a use case driven approach.*, ACM Press Series New York: ACM.
25. Jennings, N. R. (2001). An agent-based approach for building complex software systems. *Communications of the ACM*, *44*(4), 35–41.
26. Juan, T., Pearce, A., & Sterling, L. (2002). ROADMAP: Extending the GAIA methodology for complex open systems. In: *Proceedings of the first international joint conference on autonomous agents and multiagent systems: part 1* (pp. 3–10).
27. Jürjens, J. (2002). Umlsec: Extending uml for secure systems development. In: *ł UML 2002 the unified modeling language* (pp. 412–425). Berlin: Springer.

28. Misra, S., Kumar, V., & Kumar, U. (2005). Goal-oriented or scenario-based requirements engineering technique-what should a practitioner select? In: *Canadian conference on electrical and computer engineering* (pp. 2288–2292).
29. Morandini, M., Nguyen, D. C., Perini, A., Siena, A., & Susi, A. (2008). Tool-supported development with Tropos: The conference management system case study. In: *Agent-oriented software engineering VIII* (pp. 182–196). Berlin: Springer.
30. Müller, J. P., & Fischer, K. (2014). Application impact of multi-agent systems and technologies: A survey. In: *Agent-oriented software engineering* (pp. 27–53). Berlin: Springer.
31. Munroe, S., Miller, T., Belecheanu, R., Pechoucek, M., McBurney, P., & Luck, M. (2006). Crossing the agent technology chasm: Lessons, experiences and challenges in commercial applications of agents. *Knowledge Engineering Review*, *21*(4), 345.
32. Object Management Group. I. (2011). *OMG Unified Modelling Language version 2.4 (OMG UML), Superstructure*. Needham, MA: Object Management Group.
33. Padgham, L., & Winikoff, M. (2004). *Developing intelligent agent systems: A practical guide*. Chichester: Wiley.
34. Padgham, L., Thangarajah, J., & Winikoff, M. (2005). Tool support for agent development using the Prometheus methodology. In: *International conference on quality software* (pp. 383–388). IEEE.
35. Pavón, J., Gómez-Sanz, J. J., & Fuentes-Fernández, R. (2005). *The INGENIAS methodology and tools* (Chap. IX). In: Henderson-Sellers and Giorgini [22] (vol. 9, pp. 236–276).
36. Royston, P. (1995). The w-test for normality. *Applied Statistics*, *44*, 547–551.
37. Sommerville, I., & Sawyer, P. (1997). *Requirements engineering: A good practice guide*. New York: Wiley.
38. Sterling, L., & Taveter, K. (2009). *The art of agent-oriented modeling*. New York: MIT.
39. Thangarajah, J., Jayatilleke, G., & Padgham, L. (2011). Scenarios for system requirements traceability and testing. In: *The 10th international conference on autonomous agents and multiagent systems* (vol. 1, pp. 285–292).
40. Tilley, S., & Huang, S. (2003). A qualitative assessment of the efficacy of uml diagrams as a form of graphical documentation in aiding program understanding. In: *Proceedings of the 21st annual international conference on documentation* (pp. 184–191). New York: ACM.
41. Trencansky, I., & Cervenka, R. (2005). Agent modeling language (AML): A comprehensive approach to modeling mas. *Informatica*, *29*(4), 391–400.
42. Van Lamsweerde, A. (2001). Goal-oriented requirements engineering: A guided tour. In: *Proceedings of fifth IEEE international symposium on requirements engineering, 2001* (pp. 249–262).
43. van Lamsweerde, A. (2009). *Requirements Engineering: From System Goals to UML Models to Software Specifications*. New York: Wiley.
44. Winikoff, M., & Padgham, L. (2013). Agent oriented software engineering (Chap. 15). In G. Weiß (Ed.), *Multiagent Systems* (2nd ed.). Cambridge: MIT.
45. Wooldridge, M., Jennings, N. R., & Kinny, D. (2000). The Gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-agent Systems JAAMAS*, *3*(3), 285–312.
46. Yu, E. (1995). *Modelling Strategic Relationships for Process Reengineering*. PhD Thesis, Department of Computer Science, University of Toronto, Toronto.
47. Yue, T., Briand, L. C., & Labiche, Y. (2009). A use case modeling approach to facilitate the transition towards analysis models: Concepts and empirical evaluation. *Model Driven Engineering Languages and Systems* (pp. 484–498). Berlin: Springer.
48. Yue, T., Briand, L. C., & Labiche, Y. (2010). An automated approach to transform use cases into activity diagrams. *Modelling Foundations and Applications* (pp. 337–353). Berlin: Springer.
49. Zambonelli, F., Jennings, N. R., & Wooldridge, M. (2003). Developing multiagent systems: The gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *12*(3), 317–370.