Requirements Specification in The Prometheus Methodology via Activity Diagrams

(JAAMAS Extended Abstract)

Yoosef Abushark
RMIT University
Melbourne, Australia
yoosef.abushark@rmit.edu.au

John Thangarajah
RMIT University
Melbourne, Australia
johnt@rmit.edu.au

Tim Miller
University of Melbourne
Melbourne, Australia
tmiller@unimelb.edu.au

Michael Winikoff
University of Otago
Dunedin, New Zealand
michael.winikoff@otago.ac.nz

James Harland
RMIT University
Melbourne, Australia
james.harland@rmit.edu

ABSTRACT

In this work 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. The approach is general to all the methodologies that support similar notions in specifying requirements.

Keywords

AOSE Methodology; Goal-Oriented Requirements

1. INTRODUCTION

The agent-oriented software engineering field has a number of methodologies that assist developers in the development process, including the Prometheus methodology [2]. A fundamental aspect of all agent-oriented software engineering methodologies is the specification of requirements. We base our approach on Prometheus, however, it can be generalised to all the methodologies that support similar notions in specifying requirements.

We use the trading agent system [2] to illustrate this specification process. In Prometheus, the system is specified via scenarios, goals and interfaces to the environment. A scenario is similar to a use case [1] and describes a particular run of the system as a sequence of steps (Figure 1). These step types include percepts, actions or goals. Goals can be decomposed into sub-goals, using a goal-tree. There are three types of goal decompositions (only two are shown in Figure 2): 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 combination of the scenarios together with the goal trees forms part of the requirements for the system.

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, Marsella, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*

	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

Figure 1: Sale Transaction Scenario Description

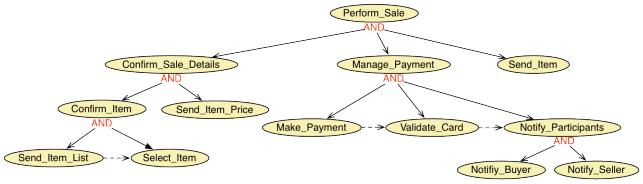


Figure 2: Goal Overview Diagram for the Trading Agent System

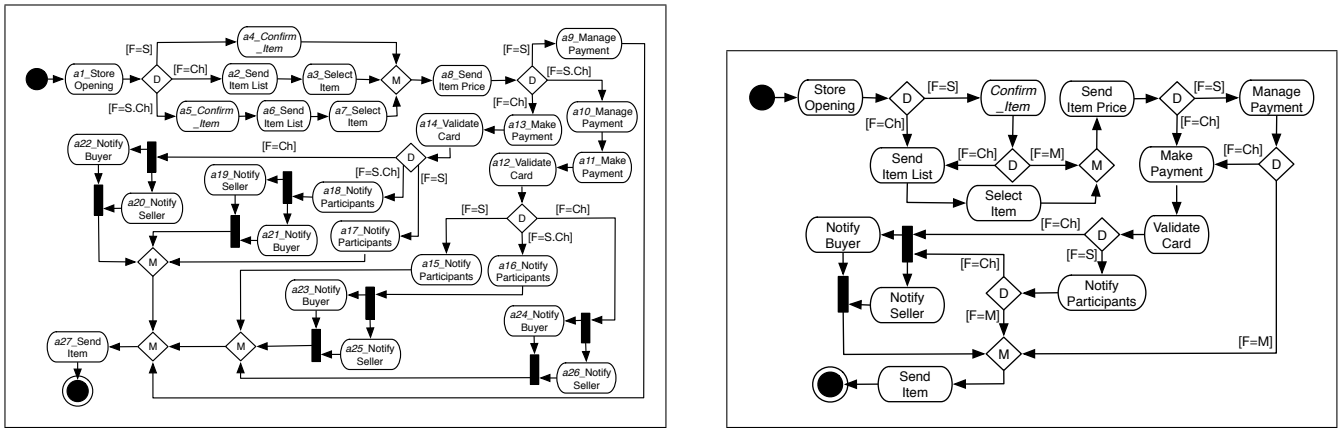
Although a scenario is a single sequence of steps there may be different ways to realise the same scenario. This is because when there are goal steps, the goals may also be realised (and hence, the requirement specified) through its children from the goal overview diagram. For example, the goal step "Notify Participants" in Figure 1 could be implemented through the step itself, the step with its children, or just the children, ("Notify Buyer" and "Notify Seller", see Figure 2).

Our approach aims to provide agent-based software designers with an automatically constructed 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.

2. METHOD

The construction process of the activity diagram of the specified scenario involves two phases:

1. *step-wise activity diagram generation*: this phase takes one step – of a given scenario – at a time, and constructs its



(a) Original Activity Diagram

(b) Reduced Activity Diagram

Figure 3: Activity diagram that merges the scenario in Figure 1 with the goal tree in Figure 2 (F:Flow, S:Goal Step, Ch: Children, P: Parent)

equivalent activity diagram structure. Such a description reflects the transformation of the steps in a given scenario into activity diagram control fragments. Actions and percepts 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. Then, the different structures are concatenated to form the complete activity diagram corresponding to the specified scenario (Figure 3a).

2. *activity diagram reduction*: the generation phase results in an activity diagram with duplicate nodes. As it is shown in Figure 3a, 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 This phase intends to reduce these duplicates, if possible, while preserving the semantics of the original activity diagram (Figure 3b).

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 applies a set of rules that merge the specified scenario with its relevant information from the goal overview diagram to generate the abstract description. Then, it uses the abstract description to generate a DOT Graph source script, which Graphviz can then use to generate a graphical depiction of the activity diagram.

3. EVALUATION

We conducted a controlled experimental evaluation to measure the usefulness of activity diagram as a complementary artefact in Prometheus. 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. A pre-evaluation questionnaire was used to measure the experience of the participants. Then, each participant was given several tasks to perform. The tasks asked participants to manipulate 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. After the tasks had been completed, participants were asked to complete a post-evaluation questionnaire, asking about their experience, and their perception of the usefulness of activity diagrams..

Table 1: Summary of the total results in both approaches (CI: confidence interval value at a confidence level of 95%)

	Activity Diagam		Non-Activity	
	Scores (out of 9)	Time (in minutes)	Scores (out of 9)	Time (in minutes)
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

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 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.

The results in Table 1 shows 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. Also, participants in our experiment unanimously agreed, through the post-evaluation questionnaire, that the inclusion of the activity diagram improved their ability to understand the requirements models. Given this, we recommend the Prometheus methodology, and indeed other AOSE methodologies, to include activity diagrams as an integrated feature in the methodology.

REFERENCES

[1] I. Jacobson. *Object-oriented software engineering: a use case driven approach*. ACM Press Series. ACM Press, 1992.

[2] L. Padgham and M. Winikoff. *Developing intelligent agent systems: A practical guide*. John Wiley & Sons, Chichester, 2004.