Towards Goal-Oriented Design of Agent Systems

Jason Khallouf*
jkhallouf@softhome.net

Michael Winikoff
RMIT University
Melbourne, Australia
winikoff@cs.rmit.edu.au

Abstract

The initial step of any software engineering methodology is to form requirements. Recently, a goal-oriented *approach to requirements has been proposed and argued to be beneficial. Goals also play a key role in the implementation of proactive software agents. However, although some agent-oriented software engineering methodologies have incorporated (aspects of) goal-oriented requirements engineering, and although they target agent platforms that provide goals as an implementation construct, none of the methodologies provide a goal-oriented design process. We present modifications to the* Prometheus *methodology which make it more goal-oriented in its design phases and report on an experimental evaluation comparing the effectiveness of the original and refined methodologies.*

1. Introduction

The initial step of any software engineering methodology is to form requirements governing the operation of the system to be developed. Recently, it has been argued that a goal-oriented approach to requirements engineering is advantageous [10]. Such an approach forms system goals based on initial functional and non-functional directives, and then elaborates and refines these goals until all directives have been broken down into goals that can be achieved by single agents[1], forming the requirements specifications. The advantages of goal-oriented requirements engineering are that using goals provides a means of incorporating non-functional goals (through the use of soft-goals) [7], facilitates requirement conflict detection and resolution [10], and aids in eliciting further goals, either by means-end reasoning (asking how/why questions) [10] or by using scenarios to help identify implicit goals [6, 9]. By ensuring that the requirements specifications achieve all high-level goals, a goal-oriented approach also allows for "a precise criterion for sufficient completeness of a requirements specification" [10]. These techniques are most evident in formal goal frameworks such as KAOS [10] and *i** [12].

Several currently proposed agent methodologies, such as Tropos [2], Prometheus [8], and MaSE [4], utilise ideas from goal-oriented requirements engineering, forming system goals in their initial phases as a base for deriving and elaborating agents. However, beyond the initial phases of requirements, the goal-oriented focus of these methodologies dissipates and does not progress into the detailed design phase where agent internals are fleshed out. None of the current agent-oriented software engineering methodologies support an entirely goal-oriented process from requirements to implementation.

Given that goals are prevalent at both the start and end points of agent design, there appears to be an opportunity to develop a methodology that is goal-oriented throughout all phases. Such a methodology should lead to better agent system designs and implementations by providing an easier design process for agent developers that more strongly guides later design stages and makes decisions more intuitive, e.g. what plans an agent should have. In this paper we present a refinement of the *Prometheus* methodology that is more goal-oriented in its design phases, and evaluate the refined methodology by designing a small agent system using both the goal-oriented and original methodologies, examining the differences between the resulting designs as well as designer feedback.

There is already much work on using goals to form and refine requirements specifications [7, 13] as well as transitioning from requirements to architectural design [1, 3, 6, 10, 11]. However, very little literature exists on the transition to a more detailed design phase where the internals of agents are elaborated.

---

* Jason was at RMIT when this research was carried out.

1 The definition of agent in this context differs from the agent-oriented context. Here, an agent is simply an "active component" that has a "choice of behaviour" [10]

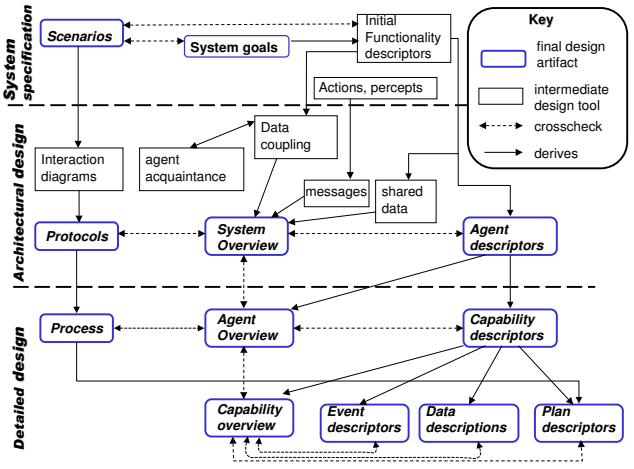

Figure 1. The Prometheus Methodology

2. The Prometheus Methodology

Prometheus [8] is an agent-oriented software engineering methodology. It is intended for use in developing any type of agent, although the final design phase targets agent platforms that utilise plans, such as BDI platforms. Prometheus consists of phases for system specification, architectural design, and detailed design, with each phase adding system entities and design artefacts. All entities feature a descriptor, containing a name, description and other fields of information pertinent to the entity type (e.g. a trigger for a plan, subgoals for a goal). Figure 1 (from [8, page 24]) shows the phases together with resultant artefacts. The description in this section is necessarily very brief, for more details see [8].

2.1. System specification

The system specification phase begins with the formation of goals. An initial list of goals, generally high-level, is identified based on the initial system description. The developer then identifies subgoals of each goal by considering how the goal can be achieved. This results in a hierarchy of goals, which are then regrouped by similarity to produce functionalities ("limited chunks of system behaviour").

Use-case scenarios are formed in tandem with this goal refinement, illustrating the steps required to achieve system goals, and possibly eliciting further goals. A use case scenario includes a sequence of steps where each step is either a goal, a percept (incoming information from the environment), an action, another scenario, or an "other" step (e.g. waiting for an event). Every scenario step also lists data used and produced, as well as the functionality involved. Figure 2 shows an example use case scenario.

Name: Scheduling a meeting
1. PERCEPT: New meeting (functionality: PP)
2. GOAL: Consider existing scheduled meetings (MP)
   Data used: Meetings
3. GOAL: Consider existing unavailability rules (PP)
   Data used: Available hours, Unavailable times
4. GOAL: Permeate meeting (IM)
5. OTHER: wait for replies (IM)
6. GOAL: Include essential members (IM)
7. GOAL: Schedule meeting (MP)
   Data produced: Meetings
8. GOAL: Notify user of meetings (PP)

Key: PP = Personal Planner, IM = Interaction Manager, MP = Meeting Planner

Figure 2. A use case scenario

2.2. Architectural design

This phase is concerned with forming the structure of the system. One of the most important tasks of this phase is to identify the agent types required within the system. This is done by grouping related functionalities into agent types in a way which attempts to reduce coupling and enhance cohesion.

Next, interaction diagrams are formed to capture messages sent and received between the newly-formed agent types. Resembling conventional sequence diagrams, interaction diagrams are usually formed from scenarios. The functionalities in each step are first replaced with their agents, and any step that is performed by an agent different to the previous step translates to a message between the agents.

Following on from interaction diagrams, interaction protocols use the Agent UML (AUML[2]) notation to precisely capture all possible agent interactions. Interaction protocols are obtained by combining related interaction diagrams and scenarios.

The overall structure of the system is captured using a system overview diagram. As its name suggests, this diagram shows all agents of the system, linked to the protocols they take part in, data they read and write, percepts and messages they receive, and actions and messages they perform/send.

2.3. Detailed design

Agents are progressively refined in terms of capabilities (nestable module-like constructs), internal events, data, and plans. *Process diagrams* (similar to UML activity diagrams,

2 http://www.auml.org

but including messages between agents) play a large role in determining what plans are necessary, and provide details on what the plan does for the plan's descriptor.

3. Refining Prometheus

Prometheus' system specification phase has a strong focus on goals. However, as the methodology proceeds into architectural design, the design focus alters from goals to messages and data, and the development of agent internals is spurred by messages and data rather than by goals. For example, interaction protocols define what messages are sent and received between agents, and these messages spur the creation of process diagrams.

In fact, the current methodology actually discards goals when formulating interaction diagrams from scenarios. Although agent descriptors do contain a listing of associated goals, the temporal context of how and when goals are achieved in the context of agents is lost. One manifestation of this loss of information is that it is hard to attribute goals to plans. Unlike the agents or capabilities that contain them, plans are temporal entities. Because of this, it is difficult to assign goals to plans working from only static information, such as goal fields in descriptors. What is needed is a temporal context for goals, showing how and when they are achieved.

Consequently, the logical approach to reasserting a goal-oriented focus in design is to maintain the presence of goals in *temporal* design phase artefacts (namely interaction diagrams, interaction protocols, and process diagrams). Since each temporal artefact is derived from a previous artefact, this refinement[3], which we term "goal derivation", supports a systematic derivation. Goal steps in scenarios are used to derive when goals are achieved in interaction diagrams, and consequently interaction protocols. In turn, these protocol goals are used to derive process diagram goals, which in turn are used in deriving the goals that a plan achieves.

The process for deriving interaction diagrams from use case scenarios is almost identical to that described in [8, section 6.1]: functionalities are replaced with the corresponding agents, messages are inserted in between scenario steps where needed and the result is expressed in the interaction diagram notation. The difference in our refined methodology is in the third step: we extend interaction diagrams to include the goals and carry goals across from the use case scenario to the interaction diagram. Goals are depicted as boxes underneath the agent's time-line.

The transition from interaction diagrams to interaction protocols involves considering all possible alternatives, drawing on alternative scenarios and the variation field of use case scenarios. Again, the key difference in our refinement is that goals are included in the interaction protocol. Figure 3 shows an example interaction protocol including goals.

Interaction protocols then form the basis for developing process diagrams. These diagrams provide a means of illustrating the sequence of activities an agent performs in response to a trigger (message or percept), and are clearly useful in determining the workings of an agent. Generally, in developing process diagrams the first received message in an interaction protocol, or percept in a scenario, is taken as the process trigger. Any goals, actions, and percepts attributed to the agent's functionalities in the scenario are added to the process diagram in chronological order, as are messages involving the agent from the interaction protocol. The addition of goals to interaction protocols aids in developing process diagrams by providing a common entity with scenarios which makes it easier to determine the relative ordering of messages (in interaction protocols) and the steps in use case scenarios. For example, the process diagram in figure 4 is for the Meeting Manager agent and was developed from the interaction protocol in figure 3. In this case there are no actions in the corresponding use case scenario, so using the refined methodology all of the information required is contained in the interaction protocol. If we were using the original Prometheus methodology then we would need to consult the use case scenario in figure 2 in order to determine which goals should be included in the process diagram. We would also need to determine the relative order of messages and scenario steps.

In developing process diagrams from interaction protocols there are two interesting situations that can occur. Firstly, in some situations a single protocol may be developed into multiple process diagrams. This occurs when the messages that are handled are not viewed as being related from the agent's perspective. For example, in Figure 3 the process triggered by the message *Meeting outcome notification* is not seen by the Personal Assistant agent as being related to earlier activities in the protocol. Secondly, it is possible for a process diagram to be based on a scenario, rather than on an interaction protocol. This occurs when a scenario is confined to a single agent, and thus no interaction protocol is necessary. In this case the goal refinement has no effect, but using scenario steps as process diagram activities ensures that process diagrams still contain goals.

Since process diagrams form the basis for plan bodies it is important to have process diagrams that are detailed and which contain goals. An inability to form adequate process diagrams leads to later difficulties in specifying the procedure field in a plan's descriptor. The ability to form detailed process diagrams is aided by the introduction of goals into interaction protocols. Thus, the proposed refinement assists

3 Two other refinements that are explored in [5] are the use of *goal coupling diagrams* instead, or in addition to, data coupling diagrams; and the introduction of additional consistency checks. These other refinements are not covered in this paper due to a lack of space.

in the development of plans. By maintaining the presence of goals throughout protocols and process diagrams, the issue of determining plan goals is addressed considerably. The subgoals of the plan are contained in the set of process diagram activities the plan carries out[4]. Figure 4 shows how a process diagram for scheduling a meeting forms the basis for two plans. It should be clear that the sequence of steps contained in the plan body of each of the two plans can be derived directly from the process diagram.

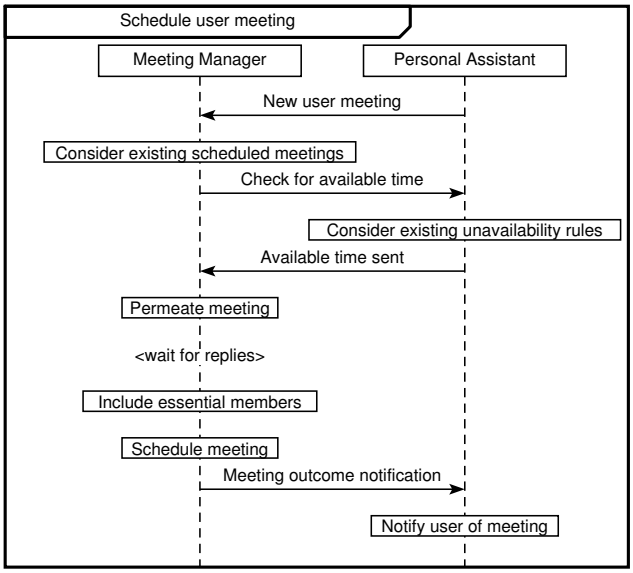

Figure 3. An interaction protocol with goals

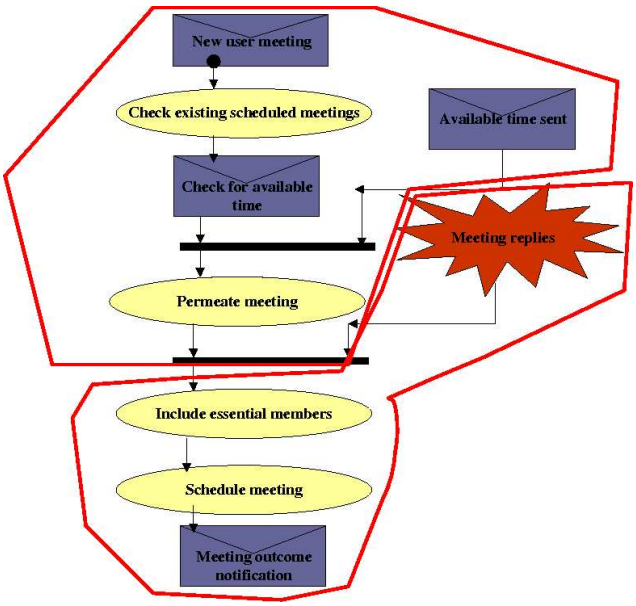

Figure 4. A Process Diagram

Designer	Design designation	Prometheus type
Author 1	R1*	Goal-oriented
Author 1	O1	Current
Author 2	O2*	Current
Author 2	R2*	Goal-oriented
Participant 1	O3	Current
Participant 2	R3	Goal-oriented
Participant 3	R4	Goal-oriented

Table 1. Designs produced for evaluation

4. Evaluation

The usability of the refined methodology, and the extent to which the refinements improve Prometheus, was evaluated by having a number of participants design an agent system using either the original version of Prometheus, the refined methodology, or both. Participants were given a completed system specification[5] for a meeting scheduling system, and were asked to complete a design. Table 1 summarises which methodology was used by the different participants. The participants "Author 1" and "Author 2" are the two authors of this paper. We use "O" and "R" to denote designs done with the "original" and "refined" methodologies. An asterisk indicates that the uncorrected system specification was used (explained below).

The meeting scheduling application had been used in 2003 as an assignment for an undergraduate class on agent oriented programming and design[6]. It involved managing a user's diary and supporting the scheduling, rescheduling, and cancellation of meetings. When (re)scheduling meetings it was possible for existing meetings to be moved in order to create a time when all essential meeting participants were free.

The evaluation proceeded in two phases. In the first phase the two authors completed (in parallel) designs using both methodologies. Author 1 used the refined methodology first, whereas Author 2 used the original methodol-

4 Where the designer has chosen not to develop a process diagram, perhaps because the process is straightforward, the plan procedure can be derived by examining the relevant interaction protocol for message and goal information, and the relevant use case scenario for actions and percepts.

5 The system specification phase is identical in the original and refined methodology, and providing a completed system specification saved time and helped make the designs more comparable.

6 We considered using student submissions for the evaluation, but the methodology used in 2003 was significantly different from the current version of Prometheus (e.g. it lacked process diagrams).

ogy first. Use of the refined methodology uncovered some minor mistakes in the system specification, and these were corrected before the second phase. In the second phase three participants, all of whom were familiar with the original Prometheus methodology, were asked to complete a design, based on the provided system specification. Participants were also provided with a description of the refined methodology (if relevant), a copy of the Prometheus Design Tool[7], a copy of the AUML protocol tool[8], and details on what deliverables were required. The participants selected for evaluating the refined methodology had also been informed of the refinements in a prior presentation.

Ideally, we hoped that each participant could finish two complete designs (one with each version of the methodology). However, this turned out to require too much time, and so each participant only did a single design, and was advised to focus their attention on agents and capabilities that were of moderate complexity in the detailed design phase. Unfortunately, even with the added design constraints only one of the participants submitted a relatively complete design (O3).

Despite the incompleteness of submitted designs the resulting designs still allow for a reasonably detailed analysis of the goal-oriented methodology in comparison to Prometheus in its current form.

At the protocol level, goal derivation indeed proved easy to do, as expected. Goals were simply kept from scenarios, and resulted in a marked improvement in readability. R1 and R2's protocols were much easier to follow than O1 and O2's, as were R3's protocols and R4's interaction diagrams. The protocols in O3 were on par with R1-R4 for readability, although this was achieved by annotating the protocol with actions.

From the overall feedback it was clear that creating process diagrams posed great difficulties for all designers, and the resulting designs substantiated this[9]. Design O2 included simple process diagrams that only contained messages sent and received from the interaction protocols, however it was reported that this diminished plan body detail. Design O3's process diagrams were comparable to R1 in detail, however the participant remarked that "creating the process diagrams and plan bodies were somewhat unguided" and "there was a lot of uncertainity (sic) for me when I was deciding what activity to put inbetween (sic) messages". As the methodology did not provide much assistance, the participant instead added activities by simply considering what internal processing the agent must do before sending and receiving messages.

These comments were echoed by the designer of R4, who also found the notation inadequate. Despite using the refined methodology, the participant also commented that prior artefacts did not provide enough information for creating process diagrams. This was unanticipated, however it appears that the participant relied solely on their interaction diagrams without interleaving with scenarios. That, coupled with the design's lack of interaction protocols, may have been the cause of this.

While design R2 did not contain process diagrams, plan bodies were noted as being easier to complete than in O2, due to goals in protocols. It is assumed that this ease in interpreting protocols and interleaving with scenarios would have translated to process diagrams, had they been created.

While R1's and R2's process diagrams benefited from goal derivation, R3's process diagram appeared to amalgamate the activities of two agents and was mostly incorrect according to the originating protocol. Given that the mistakes were fairly rudimentary, it is assumed that they were due to time shortage (as was noted by the participant) and inattentiveness, rather than difficulty in following the refinement. No feedback was given on the difficulty of forming process diagrams.

At the plan level there are two (related) issues to consider: the development of plan bodies, and the assignment of goals to plans[10]. Goal derivation offers no direct assistance in forming plan bodies, since a plan body is largely based on a given process diagram: regardless of the methodology used, when detailed process diagrams were available (as was the case with O1 and O3) plan bodies were not hard to complete. However, goal derivation does offer indirect assistance by making it easier to develop detailed process diagrams with goals.

In terms of assigning goals to plans it appears that the goal derivation refinement made it easier to assign goals. Although design O1 did adequately assign goals, this was far more difficult than with R1, as the designer had to consult scenarios in conjunction with process diagrams and protocols. Design O3 did not assign goals to agents, capabilities or plans, but the agent and capability goals were easily inferred and the participant later provided their assignment of plan goals. However, after checking with the design, it was apparent that some of the plan goals were incorrectly assigned: some goals belonged in different capabilities and even different agents. While not conclusive, this also reveals that the existing methodology does not strongly trace the assignment of goals as the design progresses to prevent these types of mistakes.

7 http://www.cs.rmit.edu.au/agents/pdt

8 http://www.cs.rmit.edu.au/~winikoff/auml

9 This is not too surprising, since little guidance is given on how to derive the activities within a process diagram [8, section 8.3]. Indeed, the process diagrams given by Padgham and Winikoff in their sample electronic book store project contain activities that do not correspond to any entity within the project.

10 Evaluating this only involved designs O1-O3 and R1-R2, since R3 and R4 did not complete the detailed design phase.

In general, goal derivation appeared to be a positive refinement. Goal-oriented interaction diagrams and protocols proved easy to produce, goal-oriented interaction protocols were often clearer to read, process diagrams were easier to produce, and consequently developing plan bodies was easier. There was also some indication that goal derivation translated into easier and more correct assignment of plan goals, however this cannot be confirmed since many of the designs did not reach the plan level.

5. Conclusion and future work

We refined Prometheus to have a stronger focus on goals in the design process by adding goals to interaction diagrams, interaction protocols, and process diagrams. This refinement was evaluated through a small case study, and appeared to be both usable and effective.

One area of future work is to extend the Prometheus Design Tool (PDT) to support interaction diagrams, interaction protocols and process diagrams. Another area of future work is to extend the evaluation: the evaluation performed, although useful, was fairly small in scale and did not allow for statistical analyses to be carried out, or for strong conclusions to be drawn. More conclusive information on the usefulness of the refined methodology could be obtained after a more extensive evaluation involving larger-scale testing, both in sample size and system size.

Acknowledgements

We would like to acknowledge the support of Agent Oriented Software Pty. Ltd. and of the Australian Research Council (ARC) under grant LP0453486. We would also like to thank Lin Padgham, David Poutakidis and Christopher Cheong for their participation during testing and evaluation, Elizabeth Haywood for her assistance and feedback during the initial stages of this project, and members of the Intelligent Agents group who provided useful feedback and discussion on this project.

References

[1] M. Brandozzi and D. E. Perry. Transforming goal oriented requirements specifications into architectural prescriptions. In *Proceedings of the First International Workshop From Software Requirements to Architectures (STRAW'01)*, Toronto, Canada, May 2001.

[2] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, May 2003.

[3] R. Darimont, E. Delor, P. Massonet, and A. van Lamsweerde. GRAIL/KAOS: An environment for goal-driven requirements engineering. In *Proceedings of ICSE-98, 20th International Conference on Software Engineering*, pages 58–62, Kyoto, Japan, April 1998.

[4] S. A. DeLoach. Analysis and design using MaSE and agentTool. In *Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2001)*, Oxford, USA, March 31 - April 1 2001.

[5] J. Khallouf. Goal-oriented design of agent systems with Prometheus, 2004. Honours thesis. Available as RMIT University School of Computer Science and IT technical report number TR-04-9.

[6] L. Liu and E. Yu. From requirements to architectural design using goals and scenarios. In *Proceedings of the First International Workshop From Software Requirements to Architectures (STRAW'01)*, Toronto, Canada, May 2001.

[7] J. Mylopolous, L. Chung, and E. Yu. From object-oriented to goal-oriented requirements analysis. *Communications of the ACM*, 18(6):483–497, June 1999.

[8] L. Padgham and M. Winikoff. *Developing Intelligent Software Agents: A Practical Guide*. John Wiley & Sons, 2004.

[9] C. Rolland, C. Souveyet, and C. B. Achour. Guiding goal modelling using scenarios. *IEEE Transaction on Software Engineering, Special Issue on Scenario Management*, pages 1055–1071, December 1998.

[10] A. van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proceedings of RE '01, IEEE International Symposium on Requirements Engineering*, pages 249–263, Toronto, Canada, August 2001.

[11] A. van Lamsweerde. From system goals to software architecture. In M. Bernardo and P. Inverardi, editors, *Formal Methods for Software Architectures*. Springer-Verlag, 2003.

[12] E. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *RE-97, 3rd International IEEE Symposium on Requirements Engineering*, pages 226–235, Annapolis, USA, January 1997.

[13] E. Yu and J. Mylopoulos. Why goal-oriented requirements engineering. In E. Dubois, A. Opdahl, and K. Pohl, editors, *Proceedings of the 4th International Workshop on Requirements Engineering: Foundations of Software Quality*, pages 15–22, Pisa, Italy, June 1998.