Goal-Oriented Design of Agent Systems: A Refinement of Prometheus and its Evaluation

Jason Khallouf and Michael Winikoff*

School of Computer Science and Information Technology
RMIT University
Melbourne, Australia
* Corresponding author
E-mail: jkhallouf@softhome.net , michael.winikoff@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 three modifications to the *Prometheus* methodology which aim to make it more goal-oriented in its design phases: we introduce a *goal coupling diagram*, as an adjunct to the existing data coupling diagram; we modify the temporal design artefacts to include goals; and we introduce a number of goal-related consistency checks. Finally, we report on an experimental evaluation comparing the effectiveness of the original and refined methodologies.

Keywords: Agent Oriented Software Engineering, Goal-Oriented Requirements Engineering, Prometheus

Reference to this paper should be made as follows: Khallouf, J. and Winikoff, M. (xxxx) 'Goal-Oriented Design of Agent Systems: A Refinement of Prometheus and its Evaluation', *International Journal of Agent-Oriented Software Engineering (IJAOSE)*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Jason Khallouf was an honours student at RMIT University in 2004, when this research was carried out. He is now working as a software developer.

Michael Winikoff is an Associate Professor at RMIT University. His research interests concern notations for specifying and constructing software. In particular, he is interested in agent oriented software engineering methodologies and is co-author of the book *Developing Intelligent Agent Systems: A Practical Guide*, published by John Wiley and sons in 2004.

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 (van Lamsweerde, 2001). Such an approach forms system goals based on initial functional and non-functional directives, and then elaborates and refines these goals until they have been broken down into goals that can be achieved by single agents[a], 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) (Mylopolous et al., 1999), facilitates requirement conflict detection and resolution (van Lamsweerde, 2001), and aids in eliciting further goals, either by means-end reasoning (asking how/why questions) (van Lamsweerde, 2001) or by using scenarios to help identify implicit goals (Liu and Yu, 2001; Rolland et al., 1998). 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" (van Lamsweerde, 2001). These techniques are most evident in formal goal frameworks such as KAOS (van Lamsweerde, 2001) and *i** (Yu, 1997).

Several currently proposed agent methodologies, such as Tropos (Bresciani et al., 2003), Prometheus (Padgham and Winikoff, 2004), and MaSE (DeLoach, 2001), 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. Although some agent-oriented methodologies use ideas from goal-oriented requirements engineering, these are used exclusively in the earlier phases of the methodologies. For example, Prometheus uses limited means-end reasoning ("how" questions) and scenarios, while Tropos, being based on the *i** framework, uses means-end reasoning, together with contribution analysis (determining whether goals have a positive or negative affect on another goal) and AND/OR decomposition to refine the system's goal structure (Giunchiglia et al., 2002).

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 (Mylopolous et al., 1999; Yu and Mylopoulos, 1998) as well as transitioning from requirements to architectural design (van Lamsweerde, 2001; Brandozzi and Perry, 2001; Liu and Yu, 2001; van Lamsweerde, 2003; Darimont et al., 1998). However, very little literature exists on the transition to a more detailed design phase where the internals of

[a]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" (van Lamsweerde, 2001)

agents are elaborated.

The focus of this paper is on refining Prometheus' *design* phases. In particular, we do not look at an early requirements phase because there is already work on agent methodologies with early requirements phases (e.g. Tropos). Also, we do not focus on implementation in this paper: the focus is rather on retaining a goal-focus when moving from a goal-oriented requirements phase ("system specification") to architectural design and detailed design.

This paper is structured as follows. We begin by reviewing the Prometheus methodology (section 2) then, in section 3, present our refinements to the methodology that make it more goal-oriented. Section 4 described an initial evaluation of the refined methodology, and we then conclude in section 5.

2 The Prometheus Methodology

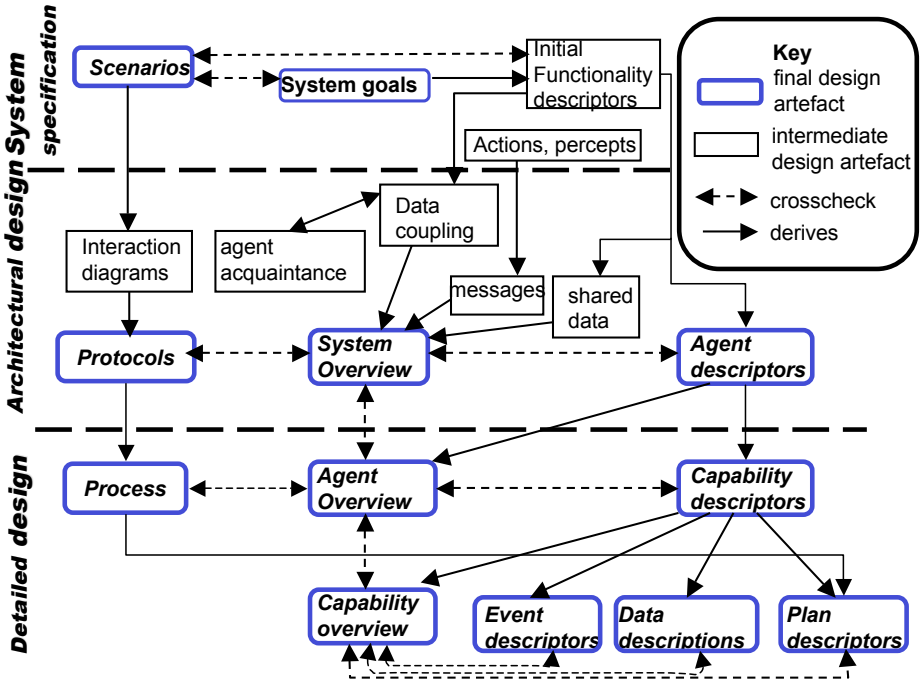

Figure 1 The Prometheus Methodology (Padgham and Winikoff, 2004, page 24)

Prometheus (Padgham and Winikoff, 2004) is an agent-oriented software engineering methodology. It is intended for use in developing any type of agent, although the final detailed 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, sub-goals for a goal). Prometheus is intended to be a pragmatic methodology that can be used by software engineers, and hence a strong focus is placed on the existence of tool support (Padgham et al., 2005).

Figure 1 shows the phases together with resultant artefacts. Although the description is linear, the intention is that the phases of the methodology be applied iteratively. The description in this section is necessarily brief, for more details see (Padgham and Winikoff, 2004). Note that the description in this paper corresponds to the version of Prometheus that was used, and is what was described by Padgham and Winikoff (2004). Prometheus has changed in a number of ways since the publication of the book, for example, (Perepletchikov and Padgham, 2005) describes a refined system specification phase, and more recently "functionalities" have been renamed to "roles".

2.1 System specification

The system specification phase begins with the formation of goals. An initial list of system goals, generally high-level, is identified based on the initial system description. The developer then identifies sub-goals 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. In this example we have the following functionalities:

- *Personal Planner* (PP) which has the goals *Update unavailable times*, *Consider existing unavailability rules*, *Notify user of meetings* and *Update available hours*.
- *Interaction Manager* (IM) which has the goals *Permeate meeting* and *Include essential members*.
- *Meeting Planner* (MP) which has the goals *Cancel meeting*, *Consider existing scheduled meetings* and *Schedule meeting*.
- *Rescheduler* which has the goals *Reschedule meeting*, *Consider meeting priority* and *Avoid cascadaing reschedules*.

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

The system specification phase also identifies the interface between the agent system and its environment. This interface is specified in terms of how the agent system affects its environment ("actions") and what information the agent system receives from its environment ("percepts").

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 (Padgham and Winikoff, 2004, page 55). Functionality names and descriptions aid in determining related functionalities, however the primary influence in determining agent groupings is a data-coupling diagram.

A data-coupling diagram, shown in Figure 3, displays all functionalities and data within the system. Data access from all functionalities can be inferred from scenarios, and is depicted as arrows joining functionalities to data. An arrow from functionality to data indicates data being produced (written), an arrow from data to functionality indicates data being used (read), and a double-ended arrow indicates data both read and written.

From the resulting diagram, functionalities (depicted as rectangles) and data sources (depicted as cylinders) are grouped together to form agents. The groups are determined by "looking for clusters of functionalities around data" (Padgham and Winikoff, 2004, page 58). Data is generally grouped together with the functionalities that produce the data. Functionalities that only use data may also be grouped with the data source if it reduces coupling and cohesion is not compromised. For the diagram in Figure 3 one possible grouping is to have one group (the "Meeting Manager" agent type) containing the three functionalities on the left and the Meetings data source, and to have a second group (the "Personal assistant" agent type) contain the Personal Planner functionality grouped with the data that it both uses and produces, i.e. the remaining two data sources.

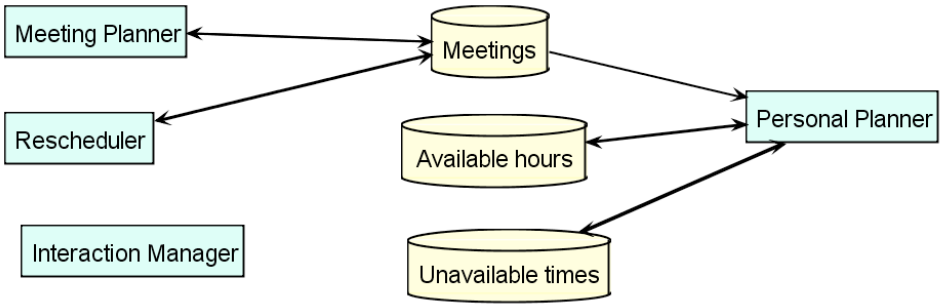

Figure 3 Data coupling diagram for a sample meeting scheduler system

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 process for forming an interaction diagram from a scenario involves firstly replacing functionalities with the agents that contain these functionalities, and secondly creating messages between any two steps that are performed by different agents. For example, given the scenario in Figure 2, step 1 is the Personal Planner functionality, which has been assigned to the Personal Assistant agent,

whereas step 2 is the Meeting Planner functionality which has been assigned to the Meeting Manager agent. Thus there is a message between step 1 and step 2. On the other hand, steps 4, 5, 6 and 7 are all done by functionalities that are contained in the Meeting Manager agent, and so no messages are needed between these steps. The interaction diagram resulting from the scenario in Figure 2 is in Figure 4.

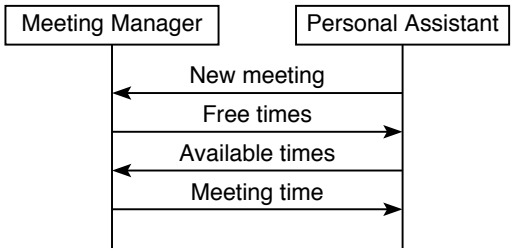

Figure 4 Interaction diagram for a sample meeting scheduler system

Following on from interaction diagrams, interaction protocols use the Agent UML (AUML[b]) notation to precisely capture all possible agent interactions. Interaction protocols are obtained by considering at each point in the interaction what alternative messages might be sent. For example, in Figure 4 the Meeting Manager responds to the *Available times* message with a *Meeting time*. However, it is also possible for the meeting scheduling to fail, so an alternative message that could be sent at this point would be a *No meeting time* message. Figure 5 shows an example AUML interaction protocol.

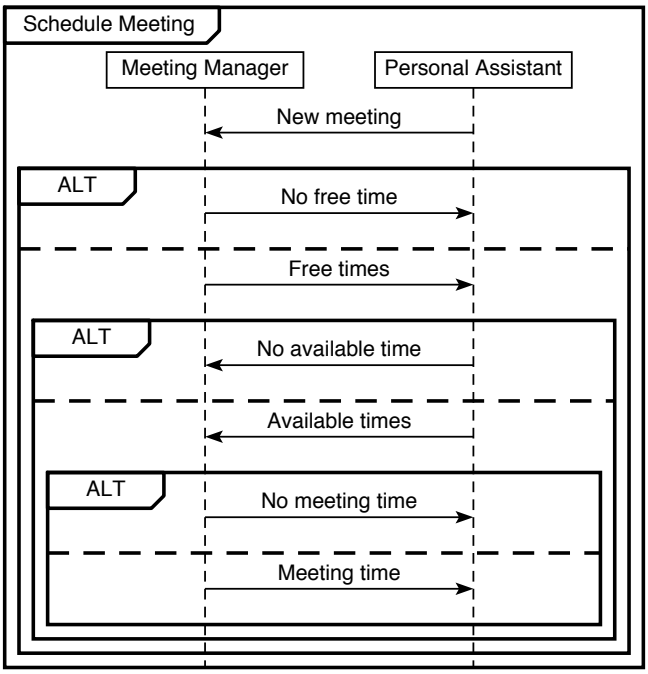

Figure 5 Interaction diagram for a sample meeting scheduler system

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

[b]http://www.auml.org

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, 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. These details are captured in a plan's procedure field.

3 Refining Prometheus

In our work we identified three areas of refinement that made Prometheus more goal-oriented (see Figure 6):

1. The use of *goal coupling diagrams* instead, or in addition to, data coupling diagrams.
2. The modification of temporal design artefacts to include goals, and using these to support systematic propagation of goals from system specification to detailed design. We term this refinement "Goal Derivation".
3. The introduction of additional consistency checks.

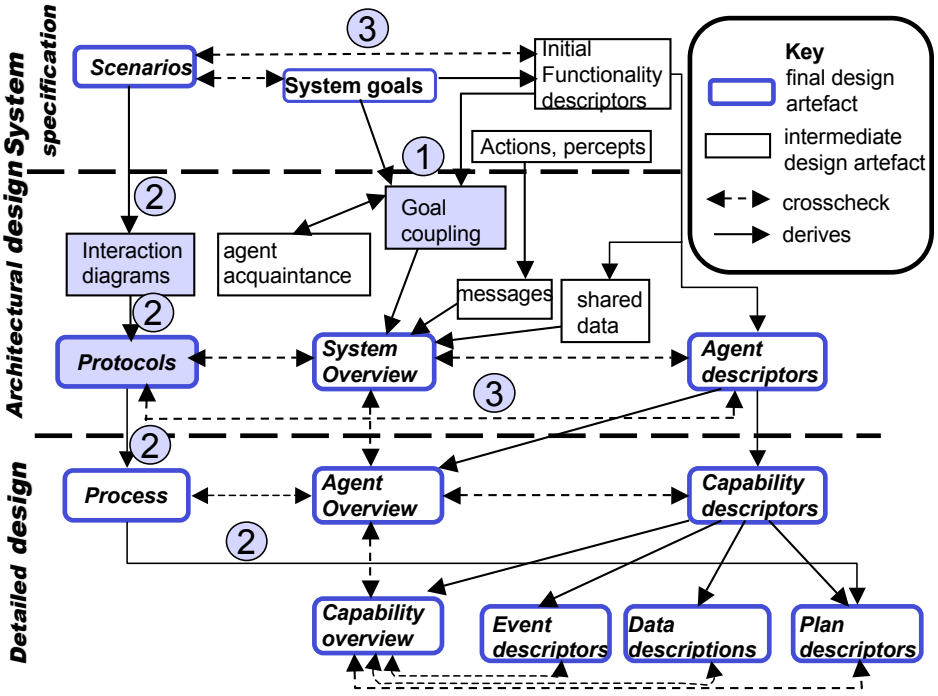

Figure 6 The Refined Prometheus methodology

Of these three refinements, the crucial one is the second, since it modified Prometheus to retain goals when moving from system specification into design. The other two refinements are less important. The first refinement attempts to make the system specification phase more goal oriented, and, as we shall see, was not entirely successful. The third refinement doesn't make the methodology any more or less goal oriented, instead it extends Prometheus' existing consistency checks, with additional consistency checks that are enabled by the second refinement.

3.1 Goal Coupling Diagram

As mentioned in section 2.2, the data-coupling diagram is the primary indicator of how functionalities should be grouped. However, as its name suggests, this diagram focusses on data and does not consider goals. Additionally, it provides no guidance for functionalities that do not use system data. Both of these drawbacks can be resolved through the use of a goal-coupling diagram.

A goal-coupling diagram (see Figure 8 for an example) consists of functionalities and top-level goals, with lines linking goals with functionalities that contribute towards that goal. The diagram is constructed by the designer firstly determining which goals should be considered to be top-level. In the case of the meeting scheduler, the top-level goals are *Schedule meeting*, *Minimise clashes*, *Form ideal schedule* and *User interaction*. Once top-level goals have been determined, the designer then in turn considers each functionality and determines towards which goals it makes a contribution. This is the most difficult step of the process, since goal contribution is somewhat subjective.

The goal coupling diagram is used to assist the designer in allocating functionalities to agents. Consequently we are interested in capturing the *significant* contributions that functionalities make towards each top-level goals, not necessarily *all* the contributions that functionalities make. Specifically, we do not consider the goals that a functionality has: if we were to consider all of the goals that are assigned to a functionality then we would obtain additional (weak) links. These additional links correspond to cases where a functionality F has goal G_S which is a sub-goal (or sub-sub-goal etc.) of a top-level goal G_T, but where F does not make a significant contribution towards G_T. For example, consider the functionality *Personal Planner*. This functionality has as one of its goals *Consider existing unavailability rules* which is a sub-sub-goal of the top-level goal *Form ideal schedule* (see figure 7). However, the functionality does not contribute significantly to this to-level goal, and so this link should not be considered as a significant constraint in the process of allocating functionalities to agents. Thus, in order to obtain a goal coupling diagram that is useful, and uncluttered with weak links, we do not aim to obtain a goal coupling diagram that shows all links, only the strong links. As discussed above, determining the strength of links, and hence which links to include in a goal coupling diagram, is somewhat subjective.

Once the goal coupling diagram has been produced, the designer uses it to identify possible groupings of functionalities into agent types. The existence of common goals that are contributed towards by a number of functionalities is evidence that the functionalities are related. On the other hand, the existence of some other goals that a functionality contributes towards which are *not* in common with related functionalities provides evidence that the functionality may not be related. For example, in Figure 8, the Meeting Planner, Interaction Manager, and Rescheduler all have a common goal (Schedule meeting) which constitutes evidence for their being related, and suggests that the three functionalities should be grouped together into an agent type. However, the Rescheduler functional-

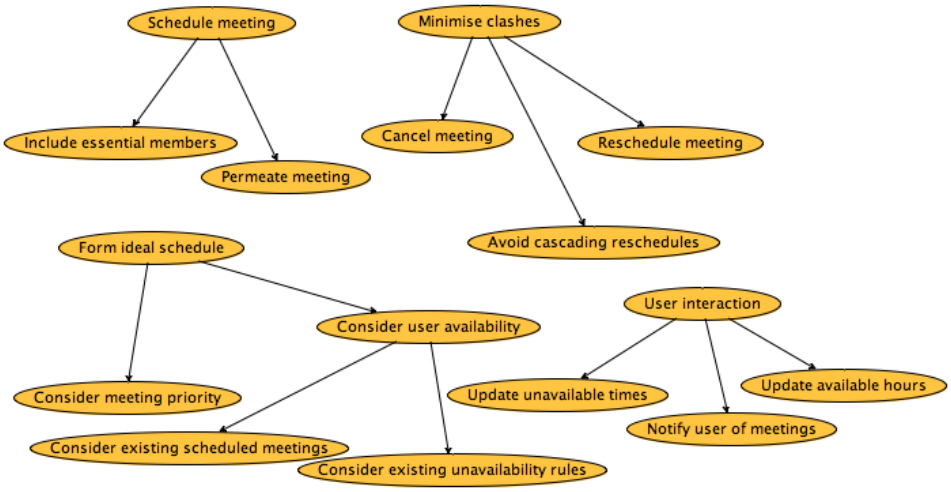

Figure 7 Goal Overview Diagram

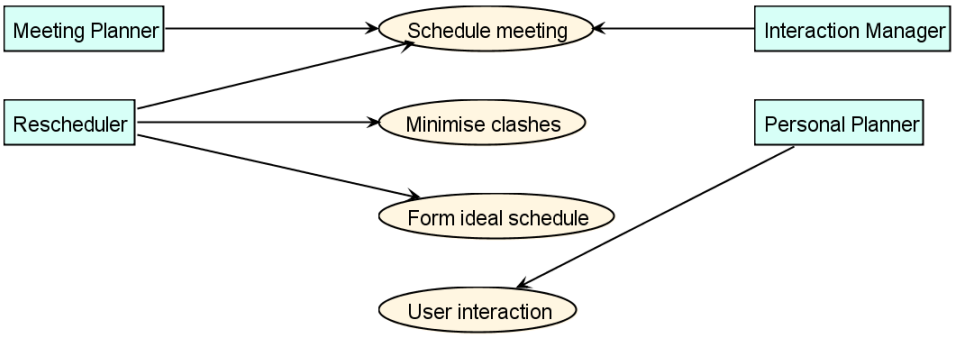

Figure 8 Goal Coupling Diagram (design R2)

ity also has two other goals that it contribute towards which are not in common with the Meeting Planner and Interaction Manager. The existence of these two other goals weakens ("dilutes") the evidence, suggesting that perhaps the designer should not group the Rescheduler functionality with the other two related functionalities.

The use of a goal coupling diagram aims to maintain the methodology's initial goal-oriented focus throughout the crucial step of agent formation by using goals to help determine agent groupings. Furthermore, since functionalities themselves are formed by grouping goals, no functionality will lack goals, and hence no functionality will have no goal contribution links. This ensures that a goal-coupling diagram provides guidance on where a functionality belongs even if the functionality does not access any data.

The drawback of a goal-coupling diagram is the difficulty in forming it, owing to its subjective nature. Deciding whether a functionality reads or writes data is a simple matter of checking use-case scenarios, and is far more clear-cut than deciding whether a functionality contributes towards the achievement of a goal. The resulting diagram may also be harder to interpret, since the links between functionalities and goals can resemble a web rather than clearly discernible segments. The reliance on functionality and goal names also

makes it crucial that the entities are carefully named.

The data-coupling diagram does not suffer from these shortcomings, and acts as a useful crosscheck. Given the correlation between high cohesion and low coupling, the resulting agent groupings from a goal-coupling diagram should be consistent with the groupings suggested by the data coupling diagram. Consequently, any major conflicts between the groupings suggested by the goal coupling and data coupling diagrams may indicate possible incorrect functionality-goal linkages in the goal-coupling diagram. It therefore seems sensible to use the data-coupling diagram in conjunction with the goal-coupling diagram, rather than supplant the data-coupling diagram altogether. It is also necessary so as to ensure that low data coupling has been maintained, which, while no longer the main focus of agent groupings, remains an important consideration.

3.2 Goal Derivation

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[c], 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, 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 by (Padgham and Winikoff, 2004, 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, where they are depicted as boxes on the agent's time-line. Note that a goal step in the use case scenario indicdates that a particular *sub-goals* is achieved. Thus, in the interaction diagram (and protocol), the presence of a goal in between messages indicates that the sub-goal is being achieved as part of the process of achieving the overall aim of the interaction.

[c] A change in focus when moving towards detailed design is normal and to be expected: the problem is that the focus on goals is lost, not that messages and data are used.

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 9 shows an example interaction protocol including goals.

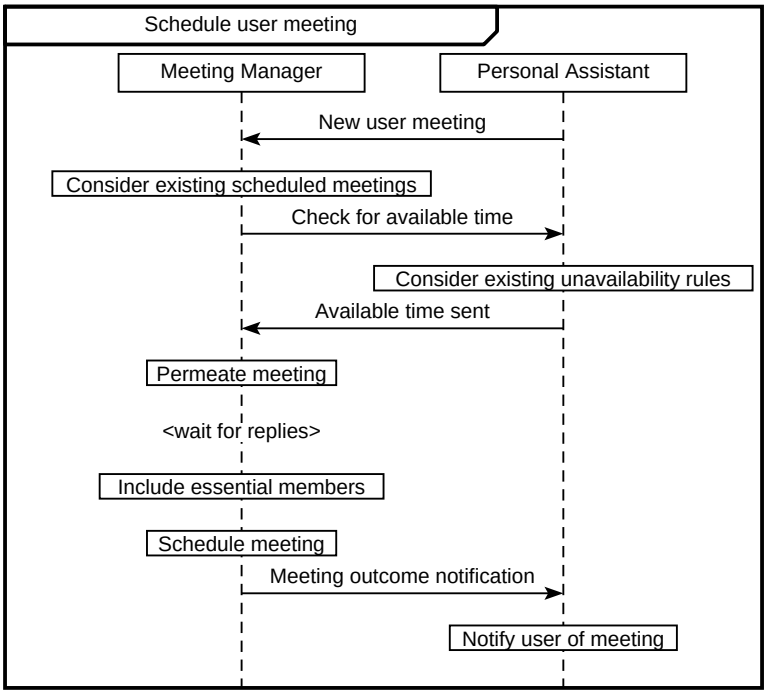

Figure 9 An interaction protocol with goals

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). 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 10 is for the Meeting Manager agent and was developed from the interaction protocol in figure 9. 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.

When 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 for a given agent. This occurs when the messages that are handled are not viewed as being related from the agent's perspective. For example, in

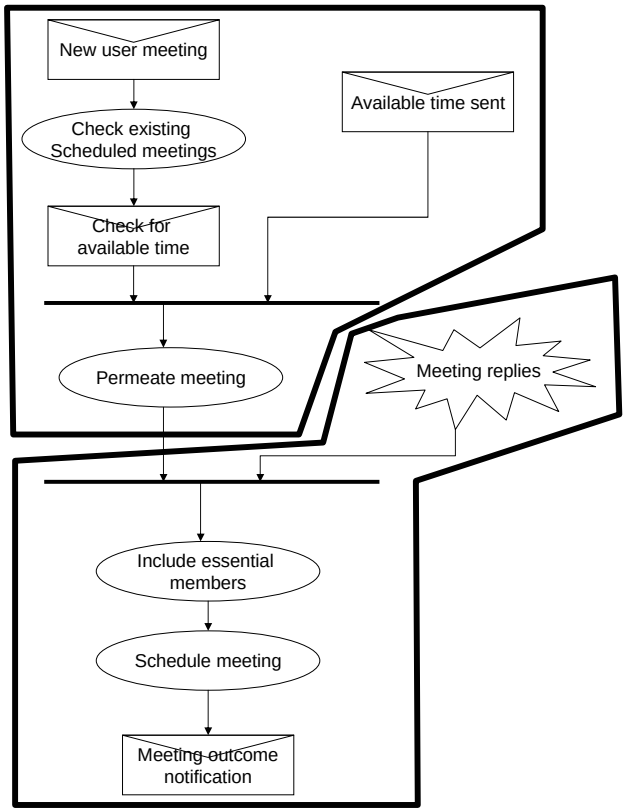

Figure 10 A Process Diagram

Figure 9 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 in the development of plans. By maintaining the presence of goals throughout protocols and process diagrams, the issue of determining plan sub-goals is addressed considerably. The sub-goals of the plan are contained in the process diagram activities that the plan carries out[d]. Figure 10 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

[d]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.

derived directly from the process diagram.

3.3 Consistency checking

Consistency checking is already present in Prometheus, and implemented in the Prometheus Design Tool (PDT) (Padgham et al., 2005). However, the derivation of goals across all artefacts introduces significant goal-related overlap between design artefacts resulting in potential points of inconsistency. For example, a protocol may indicate that a goal is assigned to a particular agent, whereas agent descriptors might indicate that the goal is assigned to a different agent. We therefore propose additional consistency checking rules. These rules, like the existing rules, assist the designer in detecting a range of errors and issues in the design as it is being developed and modified.

We have identified thirteen rules (see Figure 11 for the rules). The first nine of these are "vertical" with respect to Figure 1: they check for consistent usage of goals between design artefacts that are derived from each other, for example between functionalities and agents. The remaining four rules are "horizontal": they check for consistency between independent design artefacts at the same level, for example between protocols and agent descriptors.

Rules 1-3 ensure that a goal follows down from functionalities to agents, to capabilities, and finally to plans. These rules prevent goals from being "lost" throughout development. Rule 4 is a simple check that the designer has not accidentally assigned the goal to multiple entities. Rules 5 and 6 are derived by inverting rules 1-3.

Whereas violations of rules 1-4 are flagged as errors, violating rules 5 or 6 only raises warnings. These two rules may legitimately be violated if a developer discovers a new goal in a later phase of development. If the goal is only added at the current point of the design, the warning serves as a reminder that entities and artefacts from earlier phases should be updated to reflect the change.

Like rules 1-3, rule 7 checks that goals are propagated downwards, from scenarios to interaction diagrams/protocols. Rule 8 checks that goals in process diagrams can be traced upwards to protocols or scenarios. Checking for a correct downward succession is not feasible, since there is no guarantee that the designer has elected to complete every process diagram. Rule 9 is obtained by inverting rule 7 and gives an "upward" check from interaction protocols to scenarios. Violations of rules 7 and 8 are flagged as errors but, since it is possible for a protocol to have goals that are not in the original scenario, violations of rule 9 are merely warnings.

Note that rule 7 is only concerned with scenarios that lead to protocols. Scenarios that are confined to a single agent (which is not usually the case) will not prompt creation of a protocol, in which case rules 7 and 9 are not applicable.

Rules 10 and 11 prevent simple synchronisation mistakes, and ensure that changes to scenarios or protocols are reflected in the descriptor of the relevant functionality or agent. Rule 12 ensures that process diagrams are sufficiently detailed: the process diagram containing a plan's trigger is intended to portray the activities and goals of the plan, so a goal that is not present in the process diagram indicates an incomplete process diagram. Violations of these rules are considered to be errors. Finally, rule 13, which is an inversion of rule 10, is a warning.

Inverting rule 11 does not yield a useful rule, since it is possible for an agent goal to not be contained in any protocol. In this case, the goal should appear in a use-case scenario, and a violation will be caught by either rule 10 or 13. Inverting rule 12 is also

1. Functionality goals must be attributed to the agent which encompasses that functionality. (ERROR)
2. Agent goals must be attributed to a contained capability or plan. (ERROR)
3. Capability goals must be attributed to a contained capability or plan. (ERROR)
4. Goals must be uniquely assigned among agents and functionalities. (ERROR)
5. An agent goal should be assigned to one of the agent's functionalities. (WARNING)
6. A capability or plan goal should be assigned to its containing agent (inverse of rule 2), or capability (inverse of rule 3). (WARNING)
7. Goals in scenarios should appear in derived interaction diagrams and interaction protocols. (ERROR)
8. Goals in a process diagram should be attributed to the artefact that contains the process trigger (protocol, or a scenario if no protocol exists). (ERROR)
9. Protocol goals should be attributed to a scenario that it is developed from. (WARNING)
10. Goals attributed to a functionality in a scenario must be listed as goals in the functionality's descriptor. (ERROR)
11. Goals attributed to an agent in a protocol must be listed as goals in the agent's descriptor. (ERROR)
12. A process diagram containing a plan's trigger (either an incoming message or a percept) must also contain all of the plan's goals. (ERROR)
13. Goals listed in an functionality's descriptor should be achieved by the functionality in a scenario. (WARNING)

Figure 11 Consistency Checking Rules

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

Table 1 Designs produced for evaluation

ineffective because Prometheus does not adequately track which plans are derived from process diagrams.

4 Evaluation

The usability of the refined methodology and the extent to which the refinements improve Prometheus were evaluated by having a number of participants design an agent system using either the original version of Prometheus, the refined methodology, or both.

This evaluation was limited in scope, and thus did not allow for strong conclusions to be drawn, or for statistical analyses to be performed. However, nonetheless, the evaluation did demonstrate that the refinements were usable, and it did highlight issues with the goal-coupling diagram, as well as provide preliminary evidence that the second refinement (goal derivation) did constitute an improvement to Prometheus.

Participants were given a completed system specification[e] 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 three participants were all familiar with agents and with Prometheus. One of the participants was one of the creators of Prometheus, the other two were post-graduate students who were working in the area of agent oriented software engineering.

The meeting scheduling application had been used in 2003 as an assignment for an undergraduate class on agent oriented programming and design[f]. 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 independently completed their own designs using both methodologies. Author 1 used the refined methodology first, whereas Author 2 used the original methodology first. Use of the refined methodology uncovered some minor mistakes in the system specification, and these

[e] 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.

[f] 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).

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[g], a copy of the AUML protocol tool[h], 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 was only asked for a single design using their designated methodology, 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 three non-author participants submitted a relatively complete design (O3).

Designs R3 and R4 extended the design base into architectural design, completing goal coupling diagrams (although only R3 contained a completed data coupling diagram) and forming agents. Design R3 included only a single protocol and process diagram. Design R4 included two process diagrams but no protocols, instead including three interaction diagrams. Some attempt at completing agent descriptors was made for design R3, while R4 did feature any complete descriptors. Neither design extended into detailed design beyond creating process diagrams, so plans were not included.

From the cross-checking errors and warnings reported by PDT, it does not appear that the automated consistency checking was used by any of the participants. Also, from mistakes in the designs themselves, it does not appear that manual consistency checking was performed in designs R3 and R4.

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 original form.

4.1 Goal Coupling Diagram Evaluation

The difficulties in creating a goal coupling diagram boil down to the issue of deciding whether a functionality contributes to a given goal. The three participants used different approaches to determining this, with different results. The designer of R4 determined contribution on an abstract level, considering all that was necessary for a goal to be achieved, and then decided whether a functionality was involved in any of these activities. The problem with this approach is that most functionalities make some contribution towards the achievement of a given goal, even if it is only minor. For example, all functionalities contribute in some way to the goal Schedule meeting. When the designer realised this issue they switched to only considering what was "immediately" necessary for the achievement of a goal, which resulted in fewer links. However, even after being constrained to immediate goals, design R4 still contained nine links in its goal coupling diagram, the highest of all four designs. The designer of R2 looked for textual overlap in the names and descriptions, whereas for R3 the designer appeared to only consider entity names and did not use descriptions. The latter approach lead to obvious mistakes in R3.

[g] http://www.cs.rmit.edu.au/agents/pdt
[h] http://www.cs.rmit.edu.au/~winikoff/auml

	R1	**R2**	**R3**	**R4**
R1 (7 links total)	-	5	4	4
R2 (6 links total)	5	-	5	4
R3 (7 links total)	4	5	-	4
R4 (9 links total)	4	4	4	-

	R1	**R2**	**R3**	**R4**
R1 (4 links)	-	2	1	2
R2 (3 links)	2	-	2	2
R3 (4 links)	1	2	-	2
R4 (6 links)	2	2	2	-

Table 2 Common linkages in goal coupling diagrams

Despite these difficulties, the goal coupling diagram proved quite robust. This was assessed by looking at the degree of commonality between the different designs. Table 2 summarises how many of the links were common between the goal coupling diagrams of difference designs. Each cell contains the number of links in the row's design that were also in the column's design. The bottom table is the same as the top, but excludes links to the most-linked goal (Schedule Meeting). Table 2 (top), reveals that the diagrams shared approximately half their linkages. R1 and R2 bore the closest similarities, differing in only one functionality's linkages. R2 and R3 were also very similar, while R1 and R3 were less closely related but still similar nonetheless. R4 contained many of the linkages from the other three designs, however this was due in large part to the high total number of links contained in the diagram.

While this result appears to support the robustness of goal coupling, it bears consideration that one of the four top-level goals, 'Schedule meetings', contains a large portion of the links for each diagram. The goal contribution for 'Schedule meetings' also appeared to be easier to decide, as all designs shared at least two common linkages for this goal. In order to check if this was the case, and to determine the similarity for the remainder of the diagram, the diagrams were recounted after excluding linkages to the 'Schedule meetings' goal. The results in Table 2 (bottom) confirm that R1 and R2 are similar, as are R2 and R3, however R1 and R3 appear to far less related overall.

Despite the differences between the designs, the four diagrams generally implied the same agent groupings. Considering the relatively small size of the project, the functionalities could only be sensibly arranged into three groupings, however a critical test of the goal coupling diagram was that it did not imply unfeasible or nonsensical groupings. In this respect the diagram was almost entirely successful: the implied groupings in designs R1 and R2 were feasible, while the groupings implied by R4 matched those of the (corrected) data coupling diagram. Design R3's goal coupling diagram (Figure 13) did not imply any groupings, however removing one clearly erroneous linkage altered the diagram to clearly imply the groupings of R1 and R2. Neither of these two groupings conflicted with the data coupling diagram, although for design R2 the chosen groupings appeared to conflict slightly with the goal coupling diagram. The agent grouping suggested by goal coupling was the same as in R1, however the designer somewhat ignored this in favour of the same groupings used in R3 and R4[i]. The design base used by the designer did contain mistakes in the initial scenarios (fixed before being given to participants), and the changed data coupling appeared to be a factor in the decision, with the designer citing better cohesion and

[i] Recall that R2 was completed before R3 and R4

coupling.

As expected, the data coupling and goal coupling diagrams proved complementary. While unintended, the design base's scenarios led to a data coupling diagram where the 'Interaction Manager' functionality did not access any data. As a result, the data coupling diagram offered no information on the placement of this functionality.

By contrast R2's goal coupling diagram (Figure 8) clearly indicated that 'Interaction Manager' belonged with 'Meeting Planner'. The goal coupling diagrams in R3 and R4 contained erroneous links that weakened functionality relationships, so no clear grouping was implied for this functionality.

The data coupling diagram also served to clarify ambiguous groupings in the goal coupling diagram, by either reinforcing or diminishing weak or diluted relationships. In the case of R4 (Figure 14) a weak relationship between 'Personal Planner' and other functionalities in the goal coupling diagram did not clearly indicate its placement, while the data coupling diagram (Figure 3) clearly indicated that the functionality belonged in a separate agent.

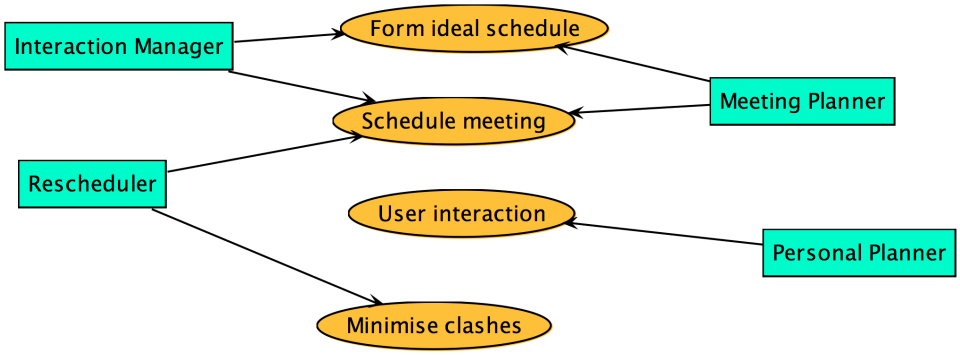

Figure 12 Goal coupling diagram for design R1

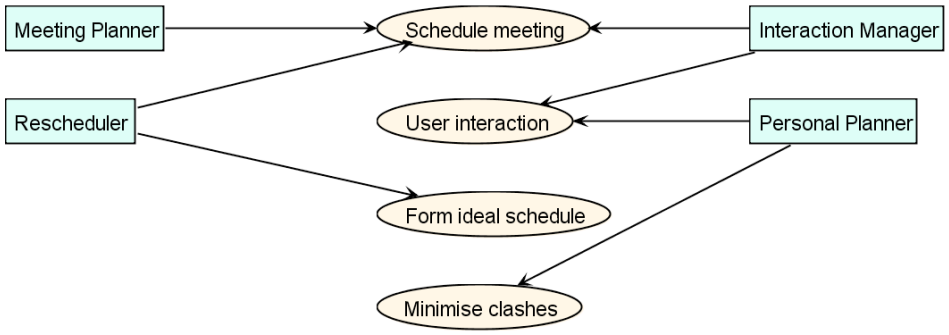

Figure 13 Goal coupling diagram for design R3

Overall, the goal coupling diagram refinement produced mixed results. In its favour, the diagram proved a useful complement to the data coupling diagram, sometimes providing agent grouping suggestions when a data coupling diagram could not. The goal grouping diagram was also unaffected by errors in scenario modelling, provided that functionalities were correctly assigned goals in their descriptors. Opposing these benefits is the diagram's difficulty: even when following the stated guidelines, determining goal contribution linkages could be difficult. When guidelines were not followed determining linkages became even more difficult and the resulting diagram far less beneficial.

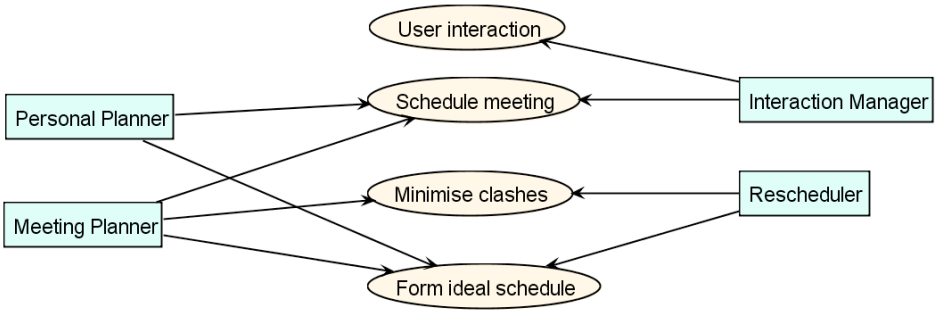

Figure 14 Goal coupling diagram for design R4

4.2 Goal Derivation Evaluation

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[j]. 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 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, designer R4 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.

On the other hand, although design R2 did not contain process diagrams, plan bodies were noted as being easier to complete than in O2, due to the presence of goals in protocols.

While R1's and R2's process diagrams benefited from goal derivation, R3's process diagram (Figure 16) appeared to amalgamate the activities of two agents and was mostly incorrect according to the originating protocol (Figure 15). 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.

[j]This is not too surprising, since little guidance is given on how to derive the activities within a process diagram (Padgham and Winikoff, 2004, 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.

At the plan level there are two (related) issues to consider: the development of plan bodies, and the assignment of goals to plans[k]. Goal derivation offers no direct assistance in forming plan bodies, since a plan body is largely based on a given process diagram. However, goal derivation does offer indirect assistance by making it easier to develop detailed process diagrams with goals. 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.

In terms of assigning goals to plans it appears that the goal derivation refinement made it easier to assign goals. Although design O1 did assign goals adequately, 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 does suggest that the existing methodology does not strongly trace the assignment of goals as the design progresses to prevent these types of mistakes.

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.

4.3 Consistency Checking Evaluation

Of the four designs using the goal-oriented methodology, only R1 appeared to use the methodology's consistency checking. As mentioned earlier, R2 did not make use of automated checking because the version of PDT used did not support it. Even so, the fact that mistakes in the original design base (detected in R1) were not detected suggested that the designer did not perform manual consistency checking either (which was confirmed by the designer).

R1 was improved by consistency checking on several occasions. Manually checking for rule 10 violations uncovered mistakes in the scenarios, where goals were being achieved by functionalities that were not originally assigned the goal. Since the goal steps in the scenario detailed when data was used or produced, incorrect assignment of functionality goals also led to an incorrect data coupling diagram.

The participants' designs also contained errors that would have been caught with consistency checking. The different naming of goals in R4's process diagram would have flagged errors due to rule 8. R3's process diagram (Figure 16) would have triggered the same error, for its inclusion of the unknown goal 'Get curr mtgs'. Similarly, an error would have been flagged for the goal 'Note failed time', which appears in the meeting scheduling protocol (Figure 15) but not in the originating scenario, nor anywhere else in the design.

Two more subtle errors in R3's process diagram identified limitations in the consistency checking rules. When viewed together with the originating protocol, it is apparent that

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

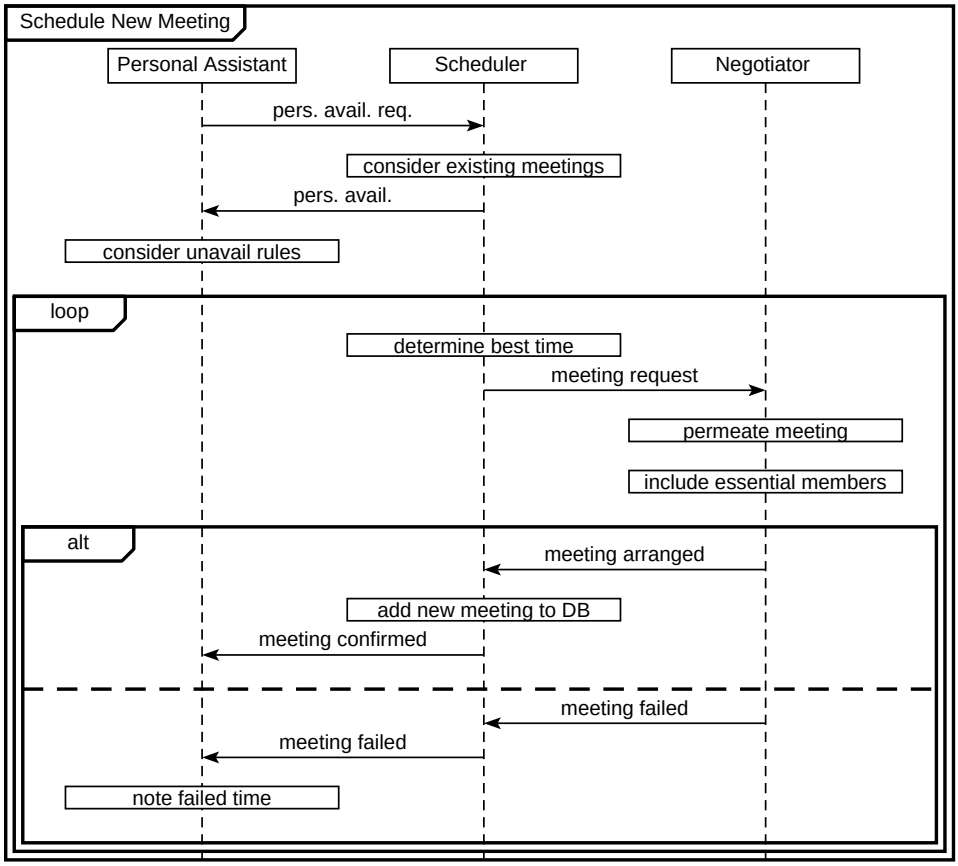

Figure 15 Interaction protocol included in design R3

the process diagram does not achieve the goal 'Consider unavailability rules' listed in the protocol, and that the goal 'Determine best time' is not achieved by the Personal Assistant, but rather by the Scheduler agent. Neither of these errors are caught by the consistency checking. Rectifying these limitations is possible by adding a rule similar to rule 7 and strengthening rule 8:

- A process diagram derived from a given protocol must contain *every* goal contained within the relevant section of the protocol.
- Goals in a process diagram should be attributed to the *same* agent in the artefact that contains the process trigger.

For the first rule, "relevant section" refers to the section of the protocol that the process diagram is detailing. This can be simply determined by considering the portion of the protocol from the process diagram's trigger to its final activity. If the final activity is the last message in the protocol, the end of the agent's timeline in the protocol should be taken as the end point. Doing so catches any missed goals after the final message, as is the case with 'Note failed time' in Figure 15. For the second rule, if the originating artefact was a scenario the designer (or tool, if performed automatically) would have to check what agent the functionality was grouped into.

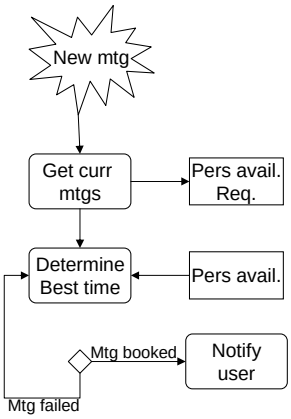

Figure 16 Process diagram for the Personal Assistant agent included in design R3 (derived from the protocol in Figure 15)

After examining the four goal-oriented designs, it becomes apparent the difference that consistency checking can make. In the cases of R3 and R4, consistency checking would have alerted the designer to errors in their process diagrams, rectifying the problem before plans were created. In the case of R2, consistency checking would have alerted the designer to scenario errors, caught in R1. This would have resolved conflicts between the goal coupling and data coupling diagrams and possibly lead to different agent groupings.

5 Conclusion

Although a number of agent-oriented software engineering methodologies have adopted ideas from goal-oriented requirements engineering, none carry the focus on goals forwards into design and detailed design. Since goals also feature at the end point of agent design — because agents are proactive — it seems sensible to try and retain the focus on goals from requirements, through design, to implementation.

We refined Prometheus to have a stronger focus on goals in the design process by:

1. Adding a goal coupling diagram.
2. Refining the derivation of interaction diagrams and interaction protocols to include goals.
3. Defining additional consistency checking rules.

These refinements were evaluated through a small case study which was limited in scope, but nonetheless provided evidence which led us to conclude that:

- There were issues in determining the contribution of functionalities to goals.
- Despite these issues, the goal coupling diagrams were somewhat robust in that they had many common linkages across designs and generally implied the same agent groupings.
- Goal derivation proved easy to do, and was beneficial, assisting in the formation of process diagrams, and hence in correctly attributing goals to plans and defining plan bodies.

- The additional consistency checking rules were useful: they were not used by most designers, but would have detected a wide range of errors and issues had they been used.

We therefore proposed that, due to issues in determining the contributions of functionalities, the goal coupling diagrams should be used in conjunction with the existing data coupling diagram, instead of replacing it. On the more positive side, the goal derivation refinement appeared to be successful in making the design process more goal-oriented, and easier.

Although the Prometheus Design Tool (PDT) was extended to support the goal coupling diagram (see Figure 17), future work remains to extend the PDT to support interaction diagrams, interaction protocols and process diagrams. It is also important to extend it to support all of the new consistency rules identified.

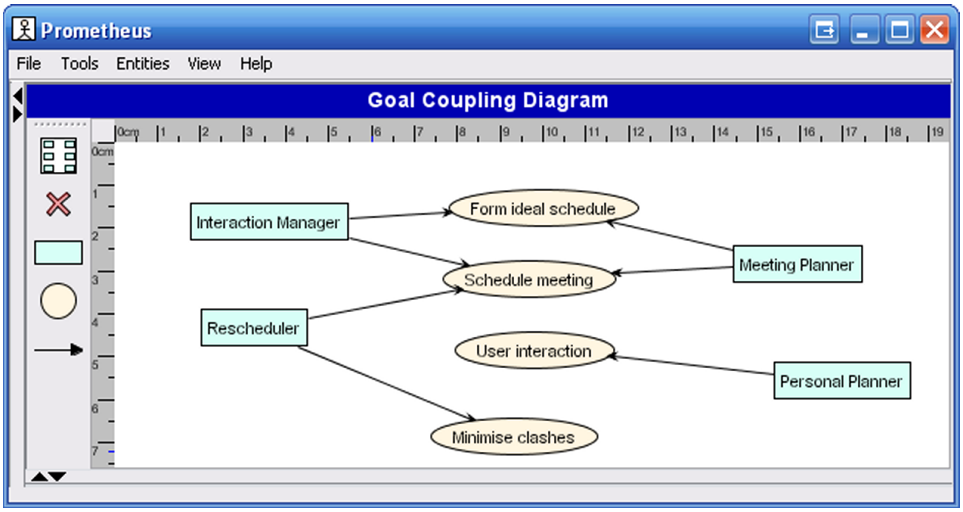

Figure 17 A Goal Coupling Diagram in the extended PDT. The navigation panes have been collapsed for space.

Another area of future work is to extend the evaluation: the evaluation performed, although useful, was 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 grants LP0453486 and LP0347025. 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

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

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

Darimont, R., Delor, E., Massonet, P., and van Lamsweerde, A. (1998). 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.

DeLoach, S. A. (2001). Analysis and design using MaSE and agentTool. In *Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2001)*, Oxford, USA.

Giunchiglia, F., Mylopoulos, J., and Perini, A. (2002). The Tropos software development methodology: Processes, models and diagrams. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 35–36. ACM Press.

Liu, L. and Yu, E. (2001). 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.

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

Padgham, L., Thangarajah, J., and Winikoff, M. (2005). Tool support for agent development using the prometheus methodology. In Cai, K.-Y., Ohnishi, A., and Lau, M., editors, *Proceedings of the Fifth International Conference on Quality Software (QSIC 2005)*, pages 383–388. Workshop on Integration of Software Engineering and Agent Technology (ISEAT).

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

Perepletchikov, M. and Padgham, L. (2005). Use case and actor driven requirements engineering: An evaluation of modifications to Prometheus. In *Multi-Agent Systems and Applications IV 4th International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS'05*.

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

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

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

Yu, E. (1997). 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.

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