The Prometheus Design Tool – A Conference Management System Case Study

Lin Padgham, John Thangarajah, and Michael Winikoff

School of Computer Science, RMIT University, Melbourne, Vic 3000, Australia
{lin.padgham,john.thangarajah,michael.winikoff}@rmit.edu.au

Abstract. This paper describes how the Prometheus Design Tool (PDT) is used to support the Prometheus methodology for designing agent systems. This is done by using an exemplar system that has been used previously in the literature, and is briefly described earlier in this volume. This paper presents the development of a design for this system using PDT. By using different tools and methodologies to design the same example system it is easier to observe the similarities and differences between both the methodologies and the tools supporting them. Prometheus and PDT, like the other systems presented in this volume, has specific strengths and features that have been developed to support the design process. However it is also evident that there is a great deal of commonality across agent methodologies that should give developers confidence that there is in fact an emerging agreed understanding as to the important aspects of designing and developing agent systems.

1 Introduction

A large number of methodologies for the analysis and design of agent systems have been proposed [1,2]. Some of the more well-known and arguably better developed methodologies include PASSI [3], MaSE [4], Tropos [5], Gaia [6], and Prometheus [7].

The fact that there are a substantial number of agent development methodologies to choose from can be confusing for developers who simply want direction in using the technology. Detailed working of a common example can provide assistance in understanding both the commonalities and the differences between approaches. This paper presents a design for a conference management system (based on the presentation of [8]), which has been developed using the *Prometheus* methodology [7] and PDT, the Prometheus Design Tool [9]. Because tools are crucial to use of a methodology for designing any real system, we focus on the design process as supported by PDT.

Key aspects of Prometheus include:

– It provides detailed guidance with specific techniques and heuristics for performing steps in the design process; PDT supports and helps structure these steps.
– It is intended to support the detailed design of agent internals as well as the structure and interactions of the agents as a system. It supports (though is not limited to) design of Belief-Desire-Intention (BDI) agents;
– It aims to strike a balance between defined structures, which support some automated reasoning, and free text or diagrams which allow designers freedom.

– It covers (to some extent), all phases of development: specification, design, implementation, and testing [10] and debugging [11];[1]
– It is designed to scale to large designs (through the use of a range of abstraction mechanisms, such as protocols and capabilities);

The methodology has been developed over more than 10 years as a result of working with industry partners who are building agent systems and agent development tools. It has also been continually refined and developed through teaching both undergraduate and postgraduate students as well as running industry seminars. The tool support has arisen out of the need to provide this at a reasonable level for building even relatively small systems. For larger systems it is essential in order to maintain consistency even of such simple things as naming. The Prometheus Design Tool (PDT) is freely available from `http://www.cs.rmit.edu.au/agents/pdt/` tool, running under Java 1.5.

The rest of this paper presents the development of the conference management system using PDT and illustrates the way in which the tool supports the different steps in the methodology. All figures in the paper are generated by PDT.

2 Prometheus Design Tool Overview

The Prometheus design methodology consists of 3 stages: system specification (which could be considered as a pre-design stage), architectural design, and detailed design. Figure 1) provides an overview of these three stages and the artifacts produced in each. In developing a Prometheus design the developer produces the design artifacts for each of these stages, which are represented in the upper left pane of PDT (see figure 2). The graphical models produced at each stage of the process are listed in this upper left pane, and are displayed and developed in the upper right pane. These are the core static models of a Prometheus design, and in figure 1 are in the centre of each stage/row.

Each entity in a Prometheus design also has a detailed descriptor with a series of fields, some of which are free text, and others of which are structured. These descriptors are displayed and developed in the bottom right pane of PDT (figure 2), and are indicated on the right hand side of each stage in figure 1. The particular descriptor on display is determined by selecting the relevant entity icon in the graphical model pane, or by selecting the entity from a list of entities in the bottom left pane (figure 2).

The left hand side of each stage in figure 1 shows design artifacts which capture models of the dynamics of a system. The primary artifacts here are *scenarios*, *protocols* and *process diagrams*. The details of scenarios and protocols are developed and displayed in pop-up windows available from the "Entities" menu at the top of the tool.[2] Icons representing these entities appear in the figures in the main graphical models pane. The notation used in PDT is shown in figure 3. The symbols used are somewhat idiosyncratic (as are those in other toolkits) and we are currently working with some of the major groups in the area, to agree a more standardised graphical notation.

[1] The testing and debugging are not yet integrated into the publically available version of PDT.
[2] Process diagrams are not currently supported.

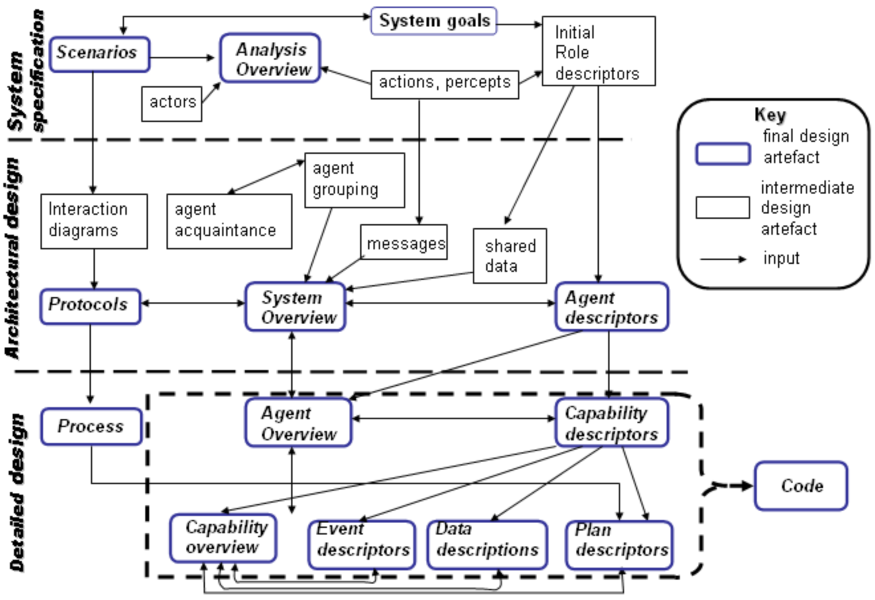
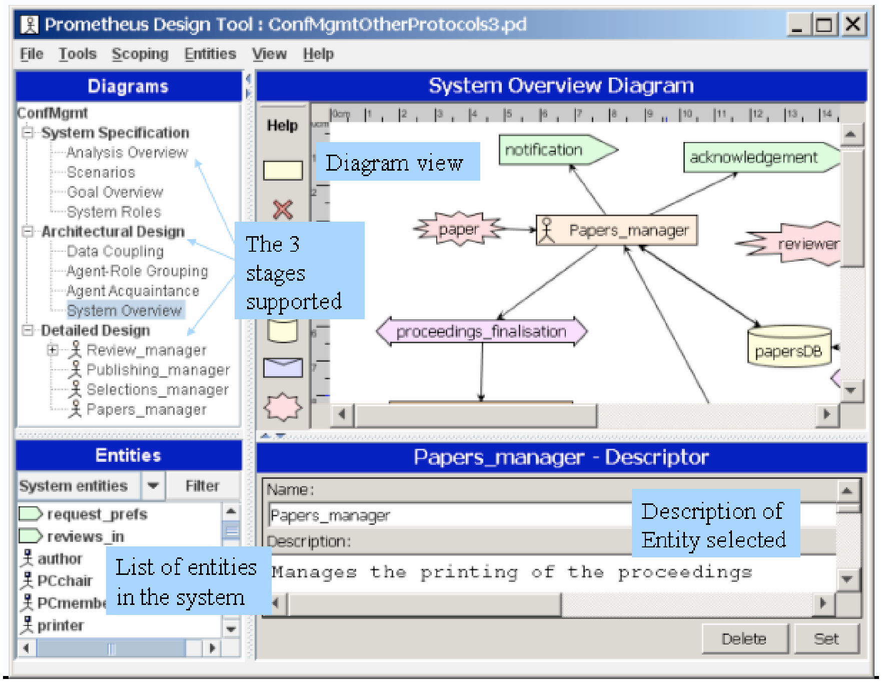


Fig. 1. Overview of Prometheus Methodology and Design Artifacts



Name:
Papers_manager
Description:
Manages the printing of the proceedings

Fig. 2. Overview of PDT

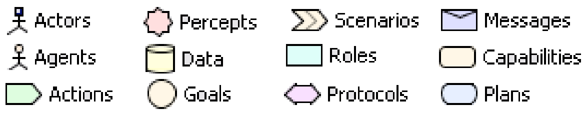


Fig. 3. Entity Notation

In the following sections we illustrate the process of design using PDT, showing the artifacts produced for the example Conference Management system. We present the design linearly covering the models in the order they are provided in PDT. However the design process is always iterative, especially within a particular stage, where typically one moves back and forth between the various models. There is also often some iteration between stages and if required, diagrams from previous stages should be revisited and revised as appropriate. One of the features of PDT is that it does enforce a degree of consistency between models, and so entities introduced in one model or stage, are often automatically propagated to other models or stages where appropriate. This assists the developer in maintaining a consistent and coherent design.

3 System Specification

It is not unusual for the initial ideas for a system to be captured very briefly, possibly in a few paragraphs. During System Specification this description must be elaborated and explored, to provide a sound basis for system design and development. In our example the Conference Management System was described as a system with four "distinct phases in which the system must operate: submission, review, decision, and final paper collection. During the submission phase, authors should be notified of paper receipt and given a paper submission number. After the deadline for submissions has passed, the program committee (PC) has to review the papers by either contacting referees and asking them to review a number of the papers, or reviewing them themselves. After the reviews are complete, a decision on accepting or rejecting each paper must be made. After the decisions are made, authors are notified of the decisions and are asked to produce a final version of their paper if it was accepted. Finally, all final copies are collected and printed in the conference proceedings." [8]. We modified this slightly to assume a structure similar to the AAMAS reviewing structure, where Program Committee members are the reviewers of papers, while Senior Program Committee members make recommendations based on the reviews.

Typically, using Prometheus, the development of the System Specification begins with identifying the external entities[3] (referred to as *actors*) that will use or interact in some way with the system, and the key *scenarios* around which interaction will occur. This is done in PDT using the 'Analysis Overview Diagram'. In figure 4 we identify Author, Printer, PCchair, PCmember and SPCmember (SPC = Senior PC) as the entities that will interact with the system. We associate them to the four main scenarios which correspond to the main functionality of the system.

[3] These may be humans or other software systems.

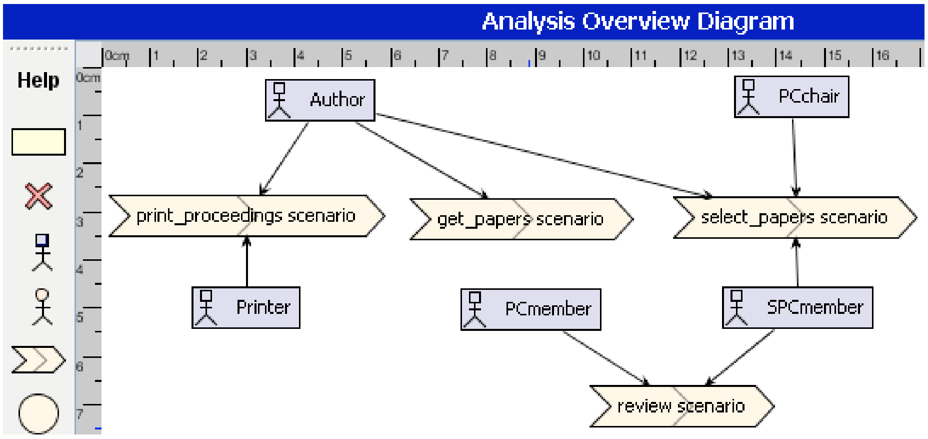


Fig. 4. Initial Analysis Overview Diagram

We then refine this diagram by identifying the *percepts* that are input to each scenario, and the *actions* produced by the system for each scenario, linking them to the appropriate actors as shown in figure 5. For example, an author submits a paper as a percept (input) to the system and the system performs an action of sending an acknowledgement back to the author. The analysis overview diagram thus defines the *interface* to the system in terms of the percepts (inputs) and actions (outputs).

The next step is to specify the details of the scenarios that we identified in the analysis overview diagram. A scenario is a sequence of *structured steps* where each step can be one of: *goal, action, percept*, or *(sub)scenario*. Each step also allows the designer to indicate the *roles* associated with that step, the *data* accessed, and a *description* of the step. These preliminary goals, roles and data that are identified are used to automatically propagate information into other aspects of the design. As steps are defined, the relevant entities are created if they do not yet exist. Figure 6 illustrates the steps of the paper reviewing scenario where the first step is a goal to invite reviewers, associated with the Review_Management role and accesses the ReviewerDB (a data structure to store reviewer details, their preferences, and paper assignments).

By default PDT creates a goal for every scenario, with the same name as the scenario. This is the goal which the scenario is intended to achieve. The name of the goal can be changed, and if desired the same goal can be associated with multiple scenarios, although this is not usually the case at the most abstract level of the Analysis Overview diagram. The goals, created from the scenarios are automatically placed into the 'Goal Overview Diagram', where goal hierarchies further describing the application are developed. For each goal, we identify its sub-goals by asking the question "how can we achieve this goal?". Figure 7 shows the goals of the conference management system. Sub-goals are either "AND"[4] or "OR"[5] branches. By default they are "AND" branches.

[4] Each sub-goal is a part of the parent goal.

[5] Sub-goals are alternative approaches to achieving the parent goal.

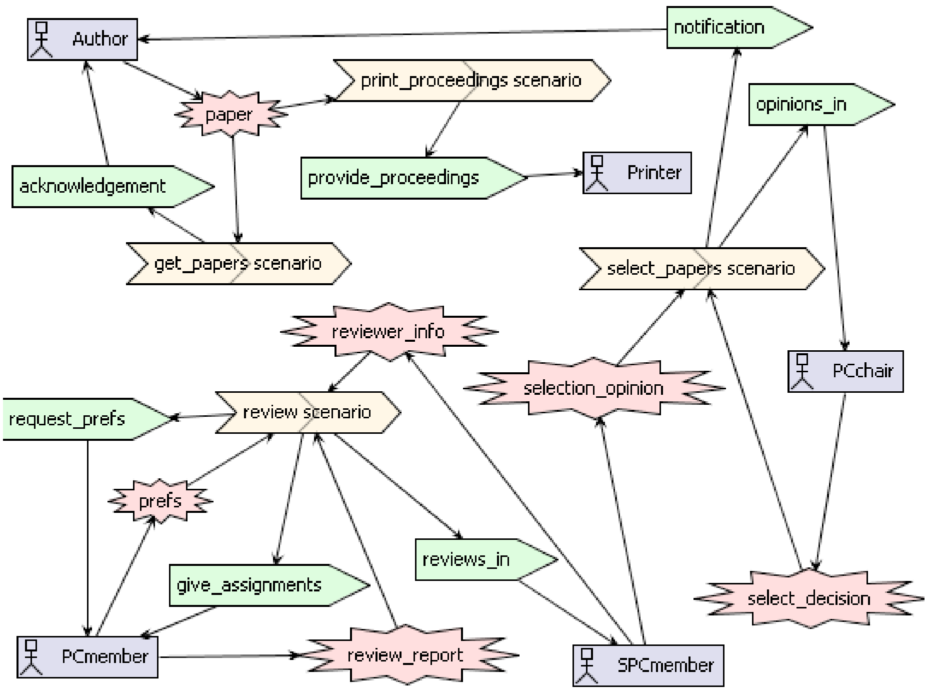
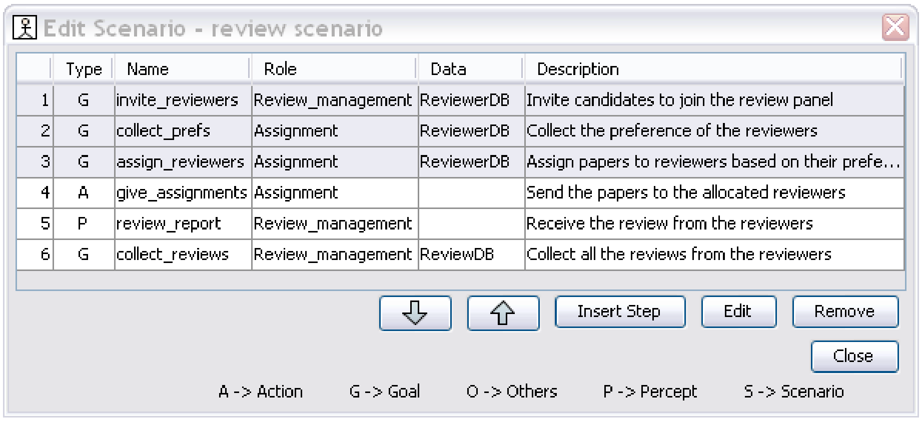


Fig. 5. Refined Analysis Overview Diagram



	Type	Name	Role	Data	Description
1	G	invite_reviewers	Review_management	ReviewerDB	Invite candidates to join the review panel
2	G	collect_prefs	Assignment	ReviewerDB	Collect the preference of the reviewers
3	G	assign_reviewers	Assignment	ReviewerDB	Assign papers to reviewers based on their prefe...
4	A	give_assignments	Assignment		Send the papers to the allocated reviewers
5	P	review_report	Review_management		Receive the review from the reviewers
6	G	collect_reviews	Review_management	ReviewDB	Collect all the reviews from the reviewers

A -> Action G -> Goal O -> Others P -> Percept S -> Scenario

Fig. 6. Scenario example - Paper Review

There is typically substantial iteration between scenario development and goal hierarchy development until the developer feels that the application is sufficiently described/defined. At this stage goals are grouped into cohesive units and assigned to *roles* which are intended as relatively small and easily specified chunks of agent

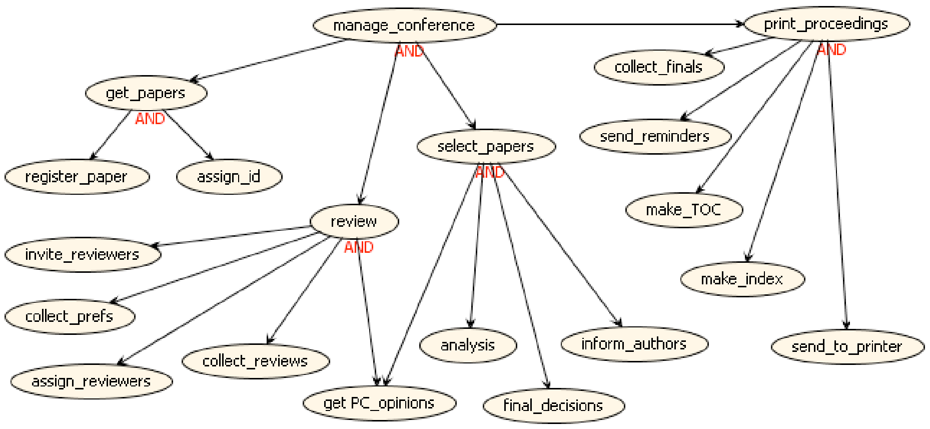


Fig. 7. Goal Overview Diagram

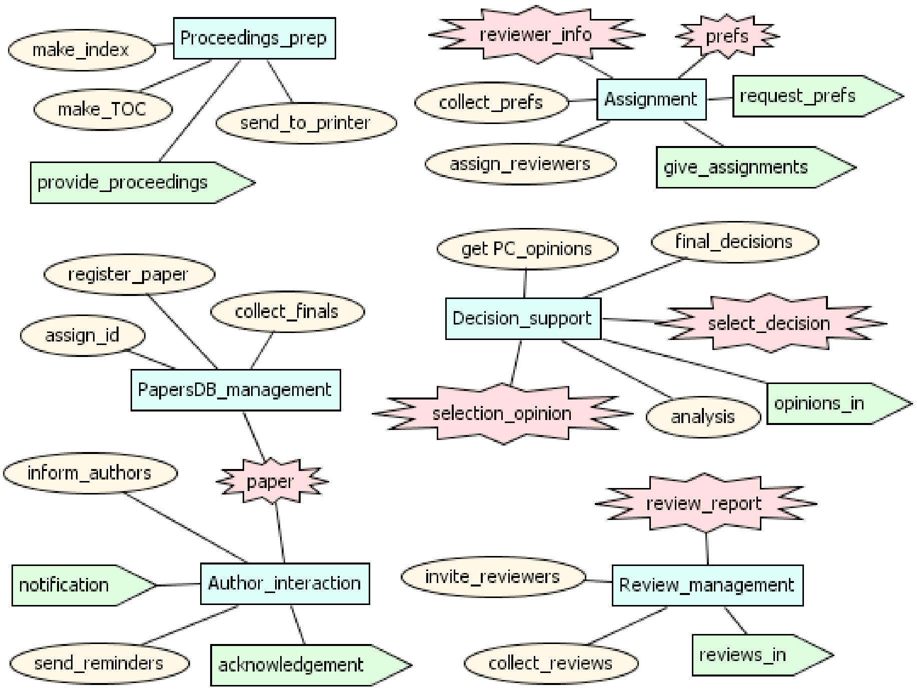


Fig. 8. System Roles Diagram

functionality. The percepts and actions are then also assigned to the roles appropriately to allow the roles to achieve their goals. This is done using the 'System Roles' diagram.

For example, Figure 8 shows that the 'Assignment' role is responsible for the goals to collect preferences (from the reviewers) and assign papers (to the reviewers). To achieve these goals the role needs the input (reviewer_info) and reviewer preferences (prefs) and should perform the actions of requesting preferences from reviewers (request_prefs) and giving out the paper assignments (give_assignments).

4 Architectural Design

The next stage is the architectural design where we specify the internal composition of the system. The main tasks here are to decide the agent types (as collections of roles) and to define the agent conversations (protocols) that will happen in order to realise the specified goals and scenarios. Decisions regarding grouping of roles into agents are captured in the 'Agent-Role Grouping Diagram'. Figure 9 shows the roles of assigning papers to reviewers (Assignment) and managing the review process (review_management) as being part of a Review_manager agent. A number of issues must be considered in determining how to group roles into agents, including standard software engineering issues of cohesion and coupling. The relationships of roles to data are also considered in determining role groupings. The Data Coupling anAgent Acquaintance diagrams can assist the designer in visualising these aspects.

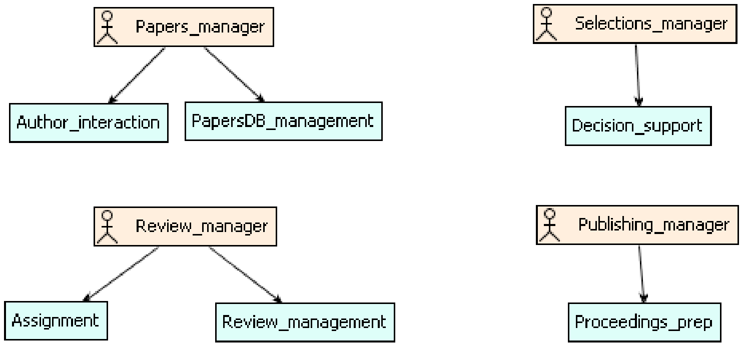


Fig. 9. Agent-Role Grouping Diagram

Once decisions have been made about how roles are grouped into agents, information can be propagated from the role specifications, to show which percepts and actions are associated with which agents. This information is automatically generated into the 'System Overview Diagram' which, when completed, provides an overview of the internal system architecture. What must be done to complete this overview is to define interactions between the agents (protocols), and to add any shared data. Figure 10 shows the system overview for our conference management system design. Observing the 'Papers_manager' agent we can see that it receives papers (percept) from authors and provides an acknowledgment (action) to them. It interacts with the 'Selections_manager' agent via the 'selection_decision' protocol to be able to send authors

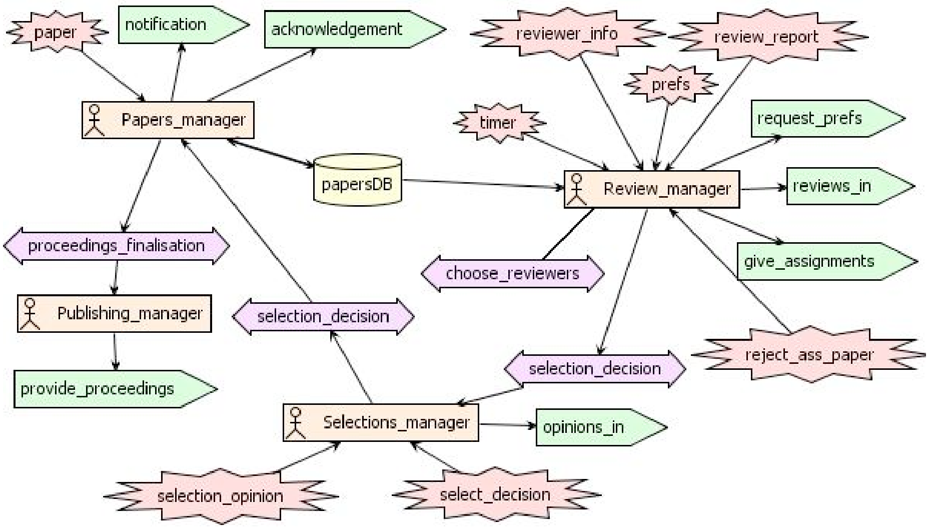


Fig. 10. System Overview Diagram

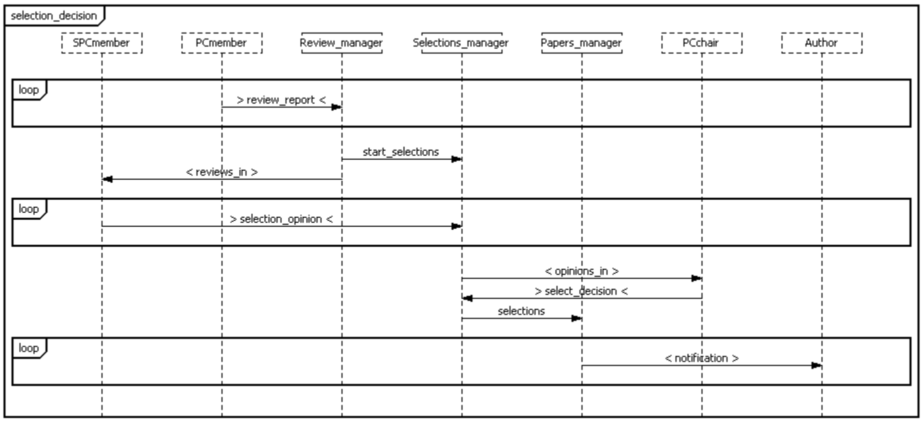


Fig. 11. Selection Decision Protocol Diagram

a notification of accept/reject (action). It also interacts with the 'Publishing_manager' agent via the 'proceedings_finalisation' protocol to provide final versions of papers to publish the proceedings.

In order to link agents with the appropriate protocols, the protocol structure must be specified using the protocol specification window (available from the Entities menu, or by double clicking the icon in the diagram). The structure of message flows is specified using a textual notation for describing a modified AUML2 [12] protocol specification. This can then be displayed as an AUML2 style figure. Any messages (or other entities) specified in the protocol, but not yet existing in the design, are created automatically.

Links are created between agents and protocol symbols, based on the specification. Prometheus modification of AUML2 allows percepts, actions and actors to be part of the protocol specification in addition to messages and agents. This often provides a better understanding of a conversation structure than showing only messages between agents.

Figure 11 shows the AUML2-like diagram of the 'selection_decision' protocol, where interactions involve three agents and four actors (identified by the dotted squares in the diagram). Percepts (which always originate with an actor and go to an agent) are written as ">percept_name<", and actions (from an agent to an actor) are written as "<action_name>". Because conversations, or protocols, do include external actors, it is possible to have a protocol connected to only one agent. An example of this in figure 10 is the choose_reviewers protocol where the review manager interacts with reviewers to give out assignments.

5 Detailed Design

The detailed design stage deals with design of the agent internals, to allow the agent to achieve the goals associated with it (via its roles and associated goals) and to engage in the interactions specified. A generic stage of detailed design describes agents in terms of capabilities, or modules. These capabilities are then finally specified in terms of plans and events, which are of necessity more specific to the implementation paradigm or platform, than the preceding steps. Specification of process diagrams is not currently supported in PDT.

The detailed design section (bottom left of figure 12) consists of a list of agent overview diagrams, one for each agent. Each agent has underneath it a list of capability overview diagrams, one for each capability included in the agent. Often the capabilities of the agent will (at least initially) correspond to the roles that were assigned to it, though roles may also be split into multiple smaller capabilities, or merged into a larger capability. For example in this case the Review_manager agent had two roles assigned to it (Assignment and review_management) and it has three capabilities: 'Reviewer registration', 'Papers assignment' and 'Review Collection'.

All the entities that were associated with the agent in the system overview diagram are propagated to the agent overview diagram, including the individual messages from protocols associated with the agent. Entities in an agent/capability overview diagram that are propagated, form part of the interface to the internals of the agent/capability and are shown as "faded" icons. These interface entities must then be connected to internal capabilities or plans defined to use or generate them. The designer needs to ensure that all the actions, percepts, messages, and data access is accounted for. For example, the 'Reviewer registration' capability handles the percept 'review_info' and modifies data in the 'ReviewerDB'.

Capabilities, which are specified using the 'Capability Overview Diagram' contain the plans which actually do things. Similarly to the agent overview diagram, percepts, messages, actions and data are propagated into this diagram and plans or (sub) capabilities are created to handle the relevant entities. A dotted line from a percept or message to a plan indicates that the percept/message is the trigger of the plan. Figure 13,

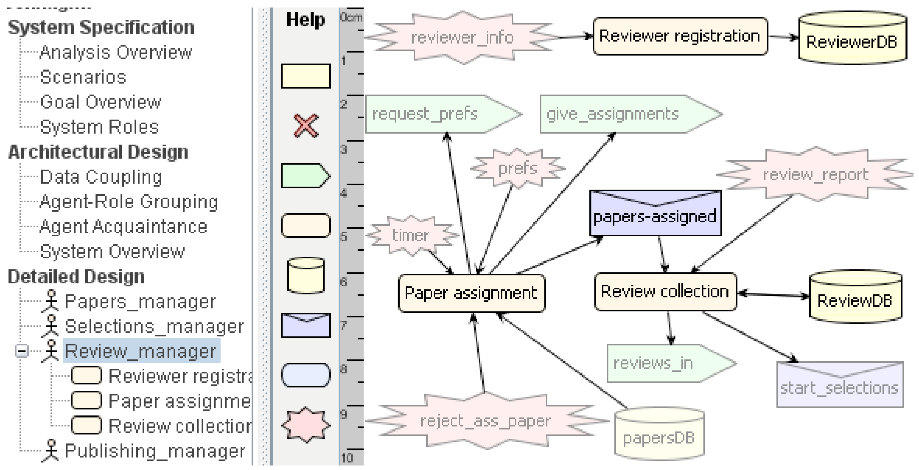


Fig. 12. Agent Overview Diagram for Reviewer_manager

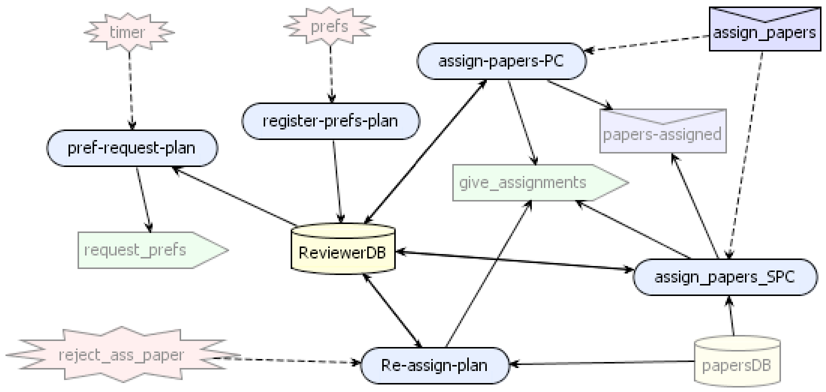


Fig. 13. Capability Overview Diagram for Paper_asignment

outlines the details of the 'Paper assignment' capability. The 'assign-papers-PC' plan is triggered by a message to assign the papers (assign_papers), reads data from 'ReviewerDB' and 'PapersDB', assigns papers to PC members (give_assignments), records the assignments in 'ReviewerDB' and, when all assignments are complete, sends a 'papers-assigned' message.

Plan descriptors allow for additional information such as a description of the plan, a context condition specifying the conditions under which this plan is applicable, a failure condition under which the plan may fail, a failure recovery procedure if the plan fails, and a description of the plan body where the developer may specify pseudocode that can be easily translated to code.

As the details of a design are developed it is very common that one recognises the need for new percepts, actions, messages, and so on. This will inevitably lead to the need to revise slightly the models developed at an earlier stage. PDT supports this by automatically introducing any new percepts and actions identified, into the system overview and analysis overview diagrams. Examples of this in the current design are the timer that is the trigger to ask reviewers to indicate which of the submitted papers they would like to review, and the reject_ass_paper which allows a reviewer to reject an assignment with which they have a conflict. These were identified during detailed design, and as a result were introduced into the System Overview, Analysis Overview and System Roles diagrams. In the System Overview the connections to the appropriate agents were also able to be made. In this case the decision was made to not leave the timer percept in the Analysis Overview or System Roles diagrams, as it did not add to understanding at System Specification level. However the reject_ass_paper does lead to a fuller understanding of the system functionality and so was connected to the review scenario and the Review_management role. The protocol choose_reviewers should then also be updated to show the role that these two new percepts play in the interaction around assigning reviewers to papers.

Once the detailed design is completed it is possible to generate skeleton (JACK) code from the Tools menu. The developer can then add to this code using a text editor. In order to maintain consistency between code and design, any additions or deletions of entities, or relationships between entities should be made in the design tool, and code regenerated on this basis. Code that is added outside that which is generated by PDT is maintained between design code iterations.

6 Features

PDT is more than simply a drawing tool for development of design diagrams. The tool maintains constraints based on the metamodel, and also provides support to prevent such simple errors as generation of unintended entities due to typographical errors. The user interface will continuously prevent the following sorts of errors:

(i) Definition: it is not possible to have references to non-existent entities, since creating a reference will create the entity if it does not exist, and when an entity is deleted all references to it are deleted as well.
(ii) Naming: it is not possible for two entities to have the same name, for example a goal and a plan both called 'assign-Papers-PC'.
(iii) Simple type errors: for example, it is not possible in PDT to connect an action and another action.
(iv) Scope constraints: for example, it is not possible to create an incoming percept to a plan without that percept also being (a) shown on the system overview diagram, and (b) shown as incoming to the agent whose plan it is.
(v) Violations of interface declarations: for example, if an agent is specified as reading a belief set, then it is not possible to create an arrow from one of the agent's plans to the belief set. Similarly, if an agent specifies that it only sends a message, then its plans cannot receive the message, and PDT does not allow the user to violate this constraint.

In addition PDT has a number of additional features available from the tools menu shown in Figure 14 Some of the additional features that the Prometheus Design Tool provides include:

– Crosschecking - this is a consistency check that is performed on demand, generating a list of errors and warnings that can be checked by the developer. Examples of a warning are writing of internal data that is never read, while an example of an error is a mismatch between the interaction protocol specified between two agents and the messages actually sent and received by processes within those agents.
– Code generation - The detailed design specification is close to code, and the tool currently provides a code generation feature that generates skeleton code of the system in the JACK agent language [13]. The skeleton code can then be completed by the developers. The tool supports repeated code generation from the design, preserving any user edited code segments.
– Report generation - One of the very useful features of the tool is its ability to generate an HTML design document. This document contains both figures and textual information, as well as an index over all the design entities. The report can also be customized such that only certain entities are included in the report. The tool can also save printable images of the various diagrams (in PNG format).
– Auto save and Backup - PDT automatically saves the current project at a set time interval (which can be changed) and also allows for creating backup files, which save the current version into a different file specified by the user.

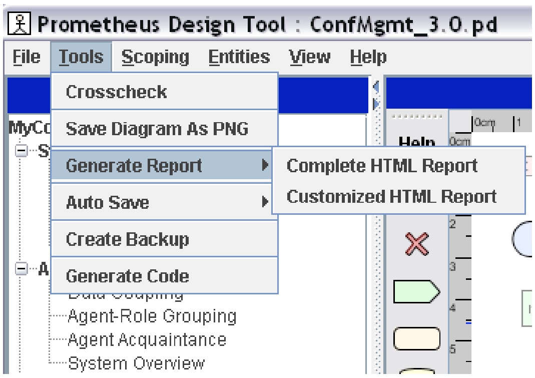


Fig. 14. Tools in PDT

PDT is also available with an Eclipse plugin, enabling it to be used within a broader IDE supporting aspects such as syntax highlighting, version management and so on. Details are available from the PDT home page at `www.cs.rmit.edu.au/agents/pdt`.

7 Future Work

There is a range of ongoing research on extending and refining the Prometheus methodology, and many aspects of this work are gradually being integrated into PDT, as they reach an appropriate stage of maturity, and as their usefulness is verified. These include:

– Testing support: we have developed automated incremental unit testing based on the design model as specified in PDT [10]. This is currently being integrated into the publically available version of PDT and should be available relatively soon.
– Debugging support: there is substantial work on interactive debugging support based on the design models specified in PDT [14,11]. This will eventually be integrated into PDT.
– Interaction development: There has been work on alternative approaches to developing agent interactions, which are more goal centric, and less message centric [15] than AUML2. This may be included into PDT as an alternative approach to interaction specification and development, which also includes detailed design of agent tasks.
– Agent organisations: We have worked with Carles Sierra on integrating an organisation design phase based on Islander, into Prometheus and PDT [16]. Facilities for integrating this design phase into PDT, using the Islander design tool, will be available soon.
– Model based production of fully executable code: A variation of PDT extends the granularity of the detailed design, as well as introducing additional specifications, in order to allow automated production of a fully functional system [17]. Aspects of this work will be incorporated into PDT soon.

In addition to incorporating ongoing research in Agent Software Engineering, into PDT, we are also continually upgrading the tool based on feedback from users. Some improvements currently in process include:

– Functionality to allow exploration of different alternatives during the design process, on “sketchpad” pages. This will make the ‘Data coupling Diagram’ and ‘Agent Acquaintance Diagram’ more useful as well as extending flexibility generally.
– Importing and exporting of existing designs to facilitate reuse and group work.
– Incorporation of existing approaches to support data design more explicitly.

We also hope to work closely with developers of other Agent Software design tools and methodologies to try and integrate our efforts to provide simpler access for industry developers and others outside the specialised community.

References

1. Henderson-Sellers, B., Giorgini, P.: Agent-Oriented Methodologies. Idea Group Publishing (2005)
2. Bergenti, F., Gleizes, M.-P., Zambonelli, F.: Methodologies and Software Engineering for Agent Systems. The Agent-Oriented Software Engineering Handbook. Kluwer Academic Publishers, Dordrecht (2004)
3. Burrafato, P., Cossentino, M.: Designing a multi-agent solution for a bookstore with the PASSI methodology. In: Proceedings of the Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002), Toronto (2002)
4. DeLoach, S.A.: Analysis and design using MaSE and agentTool. In: Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2001) (2001)

5. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi Agent Systems 8(3), 203–236 (2004)
6. Zambonelli, F., Jennings, N., Wooldridge, M.: Developing multiagent systems: the gaia methodology. ACM Transactions on Software Engineering and Methodology 12(3) (2003)
7. Padgham, L., Winikoff, M.: Developing Intelligent Agent Systems: A Practical Guide. John Wiley, Chichester (2004)
8. DeLoach, S.A.: Modeling organizational rules in the multi-agent systems engineering methodology. In: Proceedings of the 15th Canadian Conference on Artificial Intelligence, pp. 1–15 (2002)
9. Padgham, L., Thangarajah, J., Winikoff, M.: Tool support for agent development using the prometheus methodology. In: Cai, K.-Y., Ohnishi, A., Lau, M.F. (eds.) Proceedings of the Fifth International Conference on Quality Software (QSIC 2005). Workshop on Integration of Software Engineering and Agent Technology (ISEAT), sep 2005, pp. 383–388 (2005)
10. Zhang, Z., Thangarajah, J., Padgham, L.: Automated unit testing for agent systems. In: 2nd International Working Conference on Evaluation of Novel Approaches to Software Engineering (ENASE-07), pp. 10–18 (2007)
11. Poutakidis, D., Padgham, L., Winikoff, M.: Debugging multi-agent systems using design artifacts: The case of interaction protocols. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2002) (2002)
12. Winikoff, M.: Defining syntax and providing tool support for Agent UML using a textual notation. International Journal of Agent-Oriented Software Engineering 1(2), 123–144 (2007)
13. Busetta, P., Rönnquist, R., Hodgson, A., Lucas, A.: JACK Intelligent Agents - Components for Intelligent Agents in Java. Technical report, Agent Oriented Software Pty. Ltd, Melbourne, Australia (1998), `http://www.agent-software.com`
14. Padgham, L., Winikoff, M., Poutakidis, D.: Adding debugging support to the prometheus methodology. Journal of Engineering Applications in Artificial Intelligence 18(2) (2005)
15. Cheong, C., Winikoff, M.: Hermes: Implementing goal-oriented agent interactions. In: Proceedings of the Third international Workshop on Programming Multi-Agent Systems (ProMAS) (July 2005)
16. Padgham, C.S.J.T.L., Winikoff, M.: Designing institutional multi-agent systems. In: Padgham, L., Zambonelli, F. (eds.) AOSE VII / AOSE 2006. LNCS, vol. 4405, pp. 84–103. Springer, Heidelberg (2007)
17. Jayatilleke, G.B., Padgham, L., Winikoff, M.: A model driven component-based development framework for agents. Computer Systems Science & Engineering 4(20) (2005)