# Tool Support for Agent Development using the Prometheus Methodology

Lin Padgham*
*RMIT University*,
*Melbourne, Australia*

John Thangarajah
*RMIT University*,
*Melbourne, Australia*

Michael Winikoff
*RMIT University*,
*Melbourne, Australia*

## Abstract

*We believe that tool support is very important for any methodology. In this paper we describe PDT (Prometheus Design Tool) which supports the design of an intelligent agent system using the Prometheus methodology. We describe how PDT supports the various stages of Prometheus through various means such as consistency checking, support for entity propagation, and hierarchical views. We will also describe work that is currently in progress which involves the development of a plug-in for Eclipse with the aim of creating a single integrated development environment which will support the complete development cycle of an agent system from design to deployment.*

## 1. Introduction

Prometheus [6] is an intelligent agent development methodology. A key feature of this methodology is that it covers all phases of development - specification, design, implementation and testing/debugging. Like most modern software engineering methodologies, Prometheus is intended to be applied in an iterative manner. One consequence of this is that when changes are made to the design other parts of the design are often affected, and need to be updated. However, it is virtually impossible to manually ensure that the design remains consistent. Therefore some form of tool support for *consistency checking* is highly desirable. This need for tool support was also confirmed by feedback from students and others who used Prometheus prior to tool support being available.

The Prometheus Design Tool (PDT) is developed to support the Prometheus methodology. In its current state, the tool provides the system developer with a graphical user interface that supports the development of the various artefacts within the Prometheus methodology and assists in maintaining a consistent design by providing cross checking between the various diagrams, automatic propagation of design elements when possible and appropriate, and name lookup assistance. The file produced by the tool can be transformed into an agent programming language (work has been reported that transforms this output file into a Jadex agent definition file [10]). The current version of PDT is Java based and hence is platform independent. The tool is freely available at http://www.cs.rmit.edu.au/agents/pdt.

We have used PDT to develop some of our research applications and it has also been used by both undergraduate and postgraduate students to develop agent applications for assignments in a class covering agent design and programming. It has been very clear that the quality of student developed agent systems has improved substantially since the introduction of a well defined agent development methodology. Use of the methodology is far less tedious, and design documentation has been noticeably improved since the Prometheus Design Tool has been available.

PDT only supports the system specification and design stages. One of our ambitions is to develop an IDE that supports the complete development of an agent system, from specification and design, to implementation, testing and debugging. To this end, we have started developing a plug-in for Eclipse[1], which is a popular open source workbench for developing integrated tools, that uses PDT to support the design of an agent oriented system and is able to automatically generate, from the design, skeleton code of an agent programming language such as JACK [2]. The code can then be edited, compiled, tested, debugged and packaged for deployment from within Eclipse. Creating this plug-in within Eclipse allows the inheritance of the well developed features of Eclipse such as CVS integration, ANT Integration, refactoring capabilities, and many other project management features. We call this plug-in ADTP (Agent Development Tool Plug-in).

In section 2 we will provide a brief overview of the Prometheus methodology and then in section 3 describe some detail of the support that PDT provides to the method-

* Authors listed alphabetically

1 http://www.eclipse.org

ology. We then discuss our vision of ADTP in section 4 and finally discuss the current work on both PDT and ADTP and some other related tools developed by others.

## 2. Overview of the Prometheus Methodology

The Prometheus methodology contains three main phases: (i) System Specification, (ii) Architectural Design, and (iii) Detailed Design. Each of these contains a number of structured processes and results in specified design artefacts. We describe them briefly, indicating in particular the artefacts that are produced at each stage.

### 2.1. System Specification

The system specification process consists of the following steps, which are interleaved and iterated until the specification is considered sufficiently complete[2]:

- Identification of *actors* and their interactions with the system, in the form of *percepts* and *actions*;
- Developing *scenarios* illustrating the system's operation;
- Identification of the *system goals* and sub-goals;
- Identifying any *external data*;
- Grouping goals and other items into the basic *roles*[3] of the system.

Actors are any persons or roles which will interact with the system, as well as any other stakeholders whose goals should be considered. Actors may be other software systems, as well as humans. Interaction scenarios are then identified for each actor that will interact with the system, similarly to the identification of use-cases in object oriented analysis. Inputs from actor to agent system are then identified as *percepts*, while outputs from the system to actors are identified as *actions*. In some agent systems, particularly those involving interaction with a physical environment, management of percepts and actions is quite complex and requires substantial work.

Each scenario identified also generates a goal of the same name. This is similar to the Goal-Scenario coupling framework of Rolland et al. [9] which is based around the notion of a Requirement Chunk (a pair of ⟨Goal, Scenario⟩). However Prometheus uses a a unidirectional coupling, rather than the bidirectional coupling of Rolland et al. Each scenario necessarily has a goal which is linked to it (with the same name), but the more specific goals may not require a scenario.

Each identified scenario is developed with a number of detailed steps, where each step is a *goal, scenario, action* or *percept*.[4] With the coupling identified above, any nested scenario identified, automatically introduces a goal. Goals introduced as steps may warrant development of a scenario, in which case the goal step is automatically modified to be a scenario step.

Initial goals are identified via the initial scenarios as described above, and also by examining the initial system description. Further goals are then identified by a process of abstraction and refinement [11]. For each goal, we ask the question *how?* and *why?*, thus identifying new goals, and forming a goal hierarchy.

Goal refinement (according to van Lamsweerde [11], and also in Prometheus) can be of two kinds: *AND-refinement* and *OR-refinement*, If a goal is AND-refined, we mean that subgoals (or answers to the question *how?*) are steps in achieving the overall goal, and each step must be done. If it is OR-refined, then subgoals are alternative ways of achieving the goal, and doing any one of them is sufficient. Agent systems typically have both these kinds of refinements. OR-refinements allow for choice in the way of achieving goals, while AND-refinements allow for breaking down into smaller pieces. The use of AND and OR refinements also supports calculations for scoping as described in [8].

Finally, after goals and scenarios are sufficiently developed, goals are grouped into roles, where similar goals are grouped together. Actions and percepts are also allocated to roles. Scenarios are then annotated with information about which role each step belongs to, and data requirements are identified.

### 2.2. Architectural Design

The Architectural Design phase uses artefacts produced in the System Specification Phase to determine what *agent types* will be included in the system and the interaction between these agents. The steps in this phase are:

- Determine the *agent types*
- Develop the *interaction protocols*
- Develop the *system overview diagram*

Various mechanisms can be used to analyse potential groupings of roles for forming agents. These include *data coupling diagrams* and *agent acquaintance diagrams*, which assist the user in identifying agent types which are not too tightly coupled. Scenarios from the system specification can be used as a guideline to develop *interaction diagrams* which in turn can be used to develop *interaction*

2 Note that this description corresponds to subsequent work [8], not to what is described in the book [6].

3 In previous publications the term "functionalities" was used.

4 A step *other* is also allowed within Prometheus, if needed.

*protocols* which fully define the allowable agent interactions. The overall system structure is captured using a system overview diagram, which shows agent types, protocols specifying interactions between the agent types, the interface to the environment in terms of percepts, actions, and any external data. Data stores which are shared between agents are also shown.

### 2.3. Detailed Design

The Detailed Design phase uses artefacts produced in the Architectural Design Phase to define the internals of every agent in the system and to specify how agents accomplish their overall tasks. Each agent is refined in terms of its *capabilities*, *internal events*, *plans*, and *data structures*. Each capability has a capability overview diagram that captures the structure of the plans within this capability and the events that are associated with these plans. The dynamic behaviour is described by *process diagrams* based on the interaction protocols identified in the previous phase.

## 3. Prometheus Design Tool (PDT)

Figure 1, shows a sample screen-shot of PDT. As can be seen in the figure, the *Diagram* view of the left hand column in PDT provides a range of diagrams organised by the three main phases of the methodology: system specification, architectural design and detailed design.

Below the design diagrams on the left hand column is a scrollable list of all the individual entities of the system organised by entity type. This list can be filtered by entity type, for example, to only display the goals of the system. When an individual entity is selected from this list, or from any of the diagrams described above, its descriptor form is displayed on the bottom right hand column of PDT. The descriptor form describes the attributes of a given entity. For example if it is a *role* then the descriptor will have the following attributes: name of the role, a brief description, the percepts that it responds to, the actions that is produced, the information used, the information produced, and the goals associated with it.

The following subsections describe some of the features of PDT and how they support the Prometheus methodology.

### 3.1. Design Diagram Support

PDT provides the developer with a graphical user interface to enter and edit design diagrams along with appropriate descriptors for each entity.

In the *system specification* stage the *stakeholders* diagram allows adding actors, scenarios, percepts, actions, and their relative associations. This forms the top level view of the system. The developer can then develop the scenarios in the *scenarios* diagram. The *goal overview* diagram shows the system goals and their sub-goals. The *roles* diagram allows the grouping of goals, percepts and actions into roles.

The main support for the *architectural design* stage is in the *system overview* diagram. PDT maintains the consistency of this diagram and the protocol specifications with respect to interactions between agents. Interaction Protocols are not yet supported, but there is a separate prototype tool[5] which allows AUML interaction protocols to be drawn, and we are working on adding protocol support to PDT.

The *detailed design* builds the internals of each agent in terms of their capabilities, plans, events, and so on, which, when complete, can be mapped into an agent programming language such as JACK [2].

### 3.2. Consistency Checking

The Prometheus design is an iterative process and the various design stages are linked to each other. Therefore when changes are made to one aspect of the design the change may affect other aspects. It is often difficult to know all the aspects where the changes need to be propagated to, and therefore an automated process that checks for consistency between the design stages is required. Consistency checking within PDT has two aspects. One aspect that is continuously active is the user interface preventing certain errors from occurring. Some of prevented errors are:

- Definition: it is not possible to have references to non-existent entities. This is because in PDT if a reference to an entity is created then the entity is also created if it does not exist, and when an entity is deleted all references to it are deleted as well.
- Naming: it is not possible for two entities to have the same name, for example a goal and a plan both called *Determine Stocks To Buy*, nor can you have two goals with same name (duplicate entries of the same type).
- Simple type errors: when links are created between entities, only valid links are allowed. For example, it is not possible in PDT to connect an action to an action.
- Inconsistency between different levels of detail: for example, it is not possible to create an incoming percept to a plan within an agent, without that percept also being shown on the system overview diagram, as incoming to the agent whose plan it is. Similarly if an agent is specified only as reading a belief set, it cannot contain a plan which writes to that belief set.

The other aspect is a consistency check that is performed on demand, using the *Tools* menu (see Figure 1) with the *Crosscheck* option. This generates a list of errors and warnings that can be checked by the developer. Examples of

5 Available from http://www.cs.rmit.edu.au/~winikoff/auml

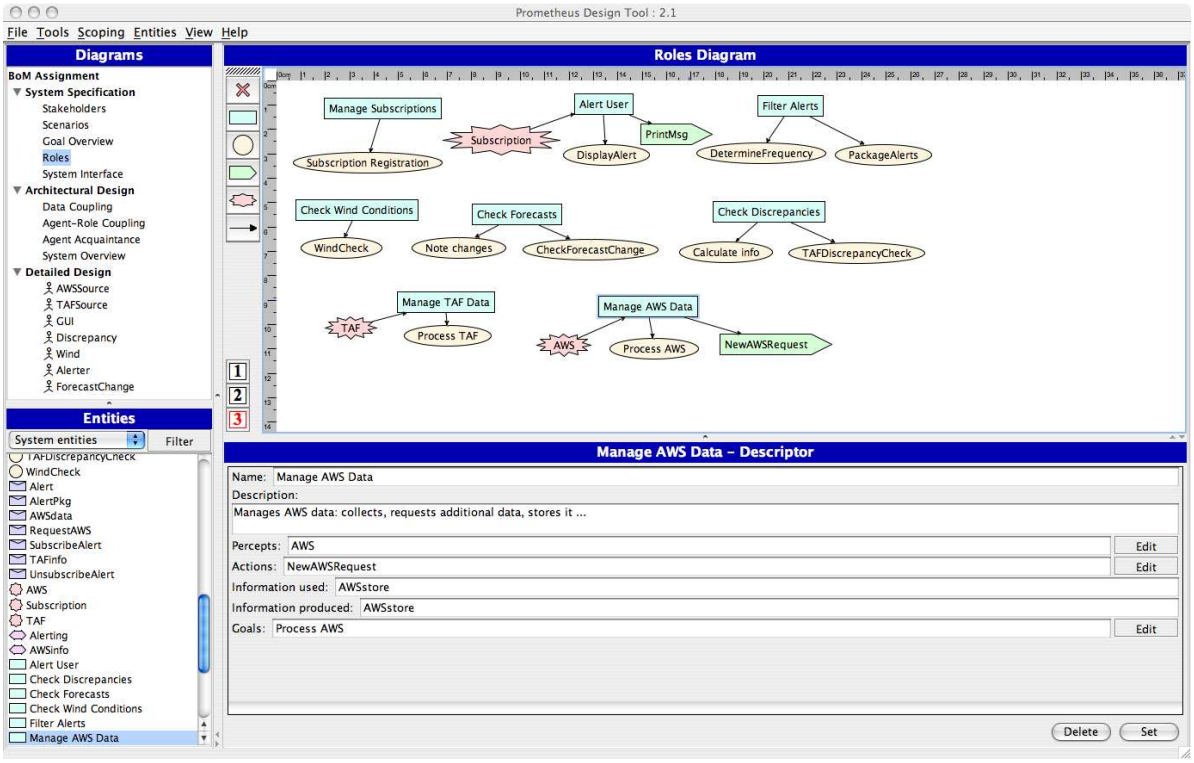


**Figure 1.** *Screen-shot of PDT.*

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. Further details on consistency checking can be found in previous work [5]. The consistency checking is based on the relationships between design artefacts documented in the Prometheus methodology, not on a meta-model.

### 3.3. Scoping

PDT also provides support for three different levels of scoping, *essential*, *conditional* and *optional*. This allows a large project to be managed in such a way that the most important aspects of the system can be fully developed, in a consistent manner, prior to starting work on less highly prioritised aspects. This aspect is described in detail in a paper also at this workshop [8].

Scoping is done based on scenarios, ranked by the developer on a five point scale, and is then propagated to other entities. The developer can also specify the relative sizes of the three partitions (levels) of the system in terms of the allowable percentage of scenarios in each partition.

When the developer requests scoping to be applied (available from the tool menu in PDT) the scenarios are partitioned in a way that is as close to the specified partition sizes as possible. All other entities are then assigned to the appropriate partition in order to maintain overall consistency. A button in the diagram window allows selection of the three different scoping partitions. Entities that are at a lower scoping level are shown faded out. This guides the developer to design prioritised entities first. Scoping can be applied or re-applied at any time during system development.

### 3.4. Entity Propagation

Whenever possible and when appropriate, information is propagated from one part of the design to another. For example, if a goal is associated with a role, and the role is associated with an agent, then the goal is automatically associated with the agent. Graphical representations of entities are also propagated. For example, if a new goal is created in the role diagram, then a graphical icon for this goal is automatically added to the goal overview diagram along with any sub goal associations that were created.

### 3.5. Hierarchical views

The tool allows for each agent to be developed with as many layers of abstraction as needed to keep each layer

manageable in size. This is achieved using capabilities and capability overview diagrams. However better support for abstraction is needed at the system level, where an improvement would be to allow a diagram which captures subsystem interaction rather than simply agent interaction.

### 3.6. Naming Lookup Assist

Entities can be added into the various diagrams by either creating new entities or by selecting ones that already exist. When selecting ones that already exist, a scrollable list of existing entities of the appropriate type is made available to the user from which to select. This ensures that no naming errors occur and is also convenient rather than having to refer back to various other diagrams. This is extremely useful when developing large systems as such trivial errors are easy to occur but difficult to debug at later stages.

### 3.7. Report Generation and Diagram Printouts

One of the useful features of PDT is its ability to generate a design document in HTML format. This document contains both figures and textual information. The textual information is obtained from the descriptors of each entity. The document also contains an index ("Dictionary") over all the design entities. PDT also allows each design diagram to be saved as an image in PNG format which can then be printed, or included into documents.

## 4. ADTP - Agent Development Tool Plug-in for Eclipse

In order to more fully support all aspects of agent software development, we have decided to integrate PDT into Eclipse[6].

Eclipse is an extremely popular, well developed, open source platform, suitable for developing commercial quality software. It provides a workbench that is fully customisable and extensible as it is built on a plug-in architecture. It provides a rich set of features for support of program development, such as: CVS integration, ANT integration, refactoring, testing and debugging (including incremental compiles and hot swapping of code), ability to incorporate plug-ins, fully integrated Java IDE, and fully customisable views and perspectives.

In order to support the ongoing improvement of our agent development workbench, we have started to build the ADTP plug-in for Eclipse. This plug in contains extension points so that it can be extended in a variety of ways. Currently PDT is incorporated into ADTP as the editor for the system design files. We[7] are in the process of building a new plug in for ADTP which will produce skeleton JACK code for the implementation of the system according to the design file produced by PDT. It will also be possible to attach alternative code production plug-ins to produce code for a variety of different agent systems.

We have taken an approach of incorporating PDT into ADTP as an external editor for the design files. We chose this approach because it allows for PDT (and other tools developed for ADTP) to also be run in a stand alone manner, without Eclipse. This can be advantageous for environments where Eclipse is not supported, or where it requires too much of the available resource capacity. However, when it is available, the extra support provided by Eclipse is invaluable in program development. Our approach also allows for PDT to be developed separately to ADTP.

Currently ADTP contains only PDT, embedded into Eclipse. However our vision for ADTP is:

- A single environment which supports all phases of software development from specification and design, through code generation, version management, testing and debugging and maintenance.
- The ability to choose different target platforms for code generation, within the same basic environment. Different design tools could also potentially be incorporated.
- An environment which maintains consistency between the design and the code. In addition to initial generation of skeleton code, this involves propagating any changes in design, through to the code base and vice versa (or at least identifying where inconsistencies have arisen).
- Provision of basic support for JACK code editing such as syntax highlighting and automatic linking to the JACK API. This is similar to, but not as rich as, the support that Eclipse provides for Java code. Since JACK is a superset of Java, the support provided by Eclipse for Java could be reused.
- Inclusion of specialised debugging tools, some of which can be based on design artefacts. We have already done work in this area [7] and are planning to incorporate this.
- Ability to compile JACK code into Java code using the JACK compiler.
- Inclusion of support tools for design of non-agent components of a system, for example a UML editor, or a database schema design tool. There is an open source UML editor plug-in available for Eclipse that could possibly be a first step in this direction.

6 http://www.eclipse.org

7 Actually a student group from the University of Melbourne are doing this under our guidance.

- Ability to build the project and package up binaries for deployment. This requires the building of both the agent and non-agent components, packaging up the appropriate binaries and resources (libraries and data files) in a manner that can be installed and run on a client system.

The general purpose features of Eclipse as already mentioned will also contribute substantially to the value of ADTP as an agent development workbench.

## 5. Discussion

There are a number of agent development toolkits, but as far as we are aware, there are relatively few which incorporate well-developed design tools such as PDT. One exception is agentTool[8], which supports the MaSE methodology [3]. However, the MaSE methodology views agents as "black boxes" and thus does not support the design of plan-based agents. The Tropos agent development methodology [1] does have some tool support[9], but this consists of a number of separate tools that cover different aspects of the software engineering process. The closest tool to PDT is TAOM which also supports a complete agent-oriented software engineering methodology (tropos [1]) that targets BDI-like agent systems. TAOM also uses Eclipse, but it does not appear to support cross checking or hierarchical views. The JACK Design Environment (JDE)[10] is also related to our work, since it provides design diagrams. However, the JDE does not support system specification activities or high level design.

An area of work that is crucial for the industrial acceptance of AOSE methodologies is the *unification* of methodologies leading to a smaller number of more widely-accepted AOSE methodologies, analogous to the emergence of the UML during the 90's. We have begun work on unifying a number of methodologies, such as Prometheus [6], ROADMAP [4], Tropos [1] and MaSE [3]. This can be expected to lead to some changes in PDT, but can also lead to incorporation of different kinds of tools within ADTP.

## Acknowledgments

This work is being supported by the Australian Research Council under grant LP0453486, in collaboration with Agent Oriented Software.

8 http://www.cis.ksu.edu/~sdeloach/ai/projects/agentTool/agentool.htm, visited 12th April 2005

9 http://trinity.dit.unitn.it/~tropos/tools.php, visited 12th April 2005

10 JDE comes as part of JACK Intelligent Agents from Agent Oriented Software. Trial and/or academic licenses are available. http://www.agent-software.com.au

## References

[1] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi Agent Systems*, 8(3):203–236, May 2004.

[2] P. Busetta, R. Rönnquist, A. Hodgson, and A. Lucas. JACK Intelligent Agents - Components for Intelligent Agents in Java. Technical report, Agent Oriented Software Pty. Ltd, Melbourne, Australia, 1998. Available from http://www.agent-software.com.

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

[4] T. Juan, A. Pearce, and L. Sterling. Roadmap: extending the gaia methodology for complex open systems. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 3–10, 2002.

[5] L. Padgham and M. Winikoff. Prometheus: A pragmatic methodology for engineering intelligent agents. In *Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies*, pages 97–108, Seattle, 2002.

[6] L. Padgham and M. Winikoff. *Developing Intelligent Agent Systems: A Practical Guide*. John Wiley and Sons, 2004. ISBN 0-470-86120-7.

[7] L. Padgham, M. Winikoff, and D. Poutakidis. Adding debugging support to the Prometheus methodology. *Engineering Applications of Artificial Intelligence*, 18(2):173–190, 2005. Special issue on Agent-oriented Software Development.

[8] M. Perepletchikov and L. Padgham. Systematic incremental development of agent systems, using prometheus. In *Proceedings of the First international workshop on Integration of Software Engineering and Agent Technology (ISEAT 2005)*, Melbourne, Australia, September 2005.

[9] C. Rolland, C. Souveyet, and B. Achour. Guiding goal modelling using scenarios. *IEEE Transactions on Software Engineering, Special Issue on Scenario Management*, 24(12):1055–1071, 1998.

[10] J. Sudeikat, L. Braubach, A. Pokahr, and W. Lamersdorf. Evaluation of agent-oriented software methodologies: Examination of the gap between modeling and platform. In P. Giorgini, J. Müller, and J. Odell, editors, *Agent Oriented Software Engineering (AOSE)*, 2004.

[11] A. van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, pages 249–263, Toronto, Canada, 2001.