Prometheus Design Tool

Lin Padgham and **John Thangarajah** and **Michael Winikoff** *
{lin.padgham,john.thangarajah,michael.winikoff}@rmit.edu.au
RMIT University, Melbourne, Australia

Abstract

The Prometheus Design Tool (PDT) supports the structured design of intelligent agent systems. It supports the Prometheus methodology, but can also be used more generally. This paper outlines the tool and some of its many features.

Introduction

The Prometheus Design Tool (PDT, see figure 1) supports the design and development of intelligent multi-agent systems, and complements the Prometheus methodology (Padgham and Winikoff 2004), although it can also be used with other approaches. The tool is written in Java and available for download at http://www.cs.rmit.edu.au/agents/pdt.

It is based on specific agent entities such as goals, plans, percepts, actions and protocols, and provides both direction and support to software engineers. It also includes automated support for testing (Zhang, Thangarajah, and Padgham 2007) and for partial code production.

It is structured around the three main design phases in Prometheus, which are:

System specification: in which the goals of the system are identified, the interface between the agents and their environment (actors) is captured in terms of actions and percepts, roles[1] are described, and detailed scenarios consisting of sequences of steps are developed.

High-level (architectural) design: in which the agent types are defined by appropriately grouping roles, the overall structure of the system is described using a system overview diagram, and interaction protocols are used to capture the dynamics of the system in terms of legal message sequences.

Detailed design: in which the internals of each agent are developed in terms of events, plans and data. These may also be encapsulated into capabilities.

PDT includes static overview models at varying levels of granularity, such as the overview of the system and its interface to the environment (analysis overview), the system overview which shows the agent types, their shared interactions and any shared data, agent overviews showing the capabilities within an agent, and capability overviews showing the detailed plans and events. PDT also contains scenario descriptions and protocol specifications which capture the dynamics of the system, and descriptors which include detailed information about each entity. PDT assists in maintaining consistency between the different levels and diagrams by both automated propagation, and by on demand consistency checking. Experience from before PDT was available confirms that this greatly reduces errors and inconsistencies in design.

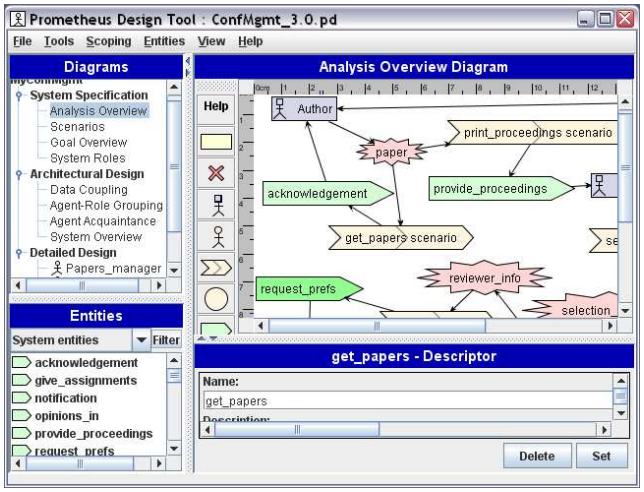

Figure 1: The Prometheus Design Tool (PDT)

Features of the Tool

PDT supports graphical development of design diagrams, but also has many other important features. Some of these are briefly outlined below:

Consistency Checking - The tool maintains constraints based on the meta-model, and also provides support to prevent such simple errors as generation of unintended entities due to typographical errors. The user interface will continuously prevent such things as:reference to non-existent entities; internal design consistency (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 dia-

*Authors in alphabetical order.

[1]Called functionalities in (Padgham and Winikoff 2004).

gram, and (b) shown as incoming to the agent whose plan it is); errors such as incorrect relationships between entity types; violations of interface declarations (e.g. 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 indicating writing). In addition to the above run time check there is also an on-demand check feature that generates a list of errors and warnings that can be checked by the developer. Further details are available in (Padgham, Thangarajah, and Winikoff 2005).

Graphical interface plus structured textual descriptors - The tool provides a direct manipulation graphical interface for creating the key diagrams of the Prometheus methodology. There are also textual forms for entity descriptors which allow a combination of free text and entries based on menus of items.

Propagation - Wherever possible, information is propagated from one part of the design to another. For example, if goals are associated with a role, and the role is associated with an agent, then the goals are also automatically associated with that agent. Similarly, graphical icons representing things that should be included in a particular diagram are automatically placed into that diagram. Whenever information about an application entity is changed, it is propagated to all relevant places in the design.

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.[2]

Report generation - A much appreciated practical feature of the tool is its ability to generate an (customizable) HTML design document. This document contains both figures and textual information, as well as an index over all the design entities. The tool can also save printable images of the various diagrams (in PNG format).

AUML protocol specification - The designer may specify AUML interaction protocols using textual notation via the protocol editor in PDT. Specification in the protocol definition also introduces entities into the other relevant models.

Code generation - PDT currently provides a code generation feature that generates skeleton code in the JACK agent language (www.agent-software.com.au). This can then be completed by the developers. The tool supports repeated code generation from the design, preserving any user edited code segments.

Automated Testing - PDT supports automated generation and execution of unit tests for testing of events, beliefs and plans, based on the design model. User defined tests can be added to those automatically generated. A test report is generated as a hyperlinked html document.

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.

[2] A useful addition would be to also allow named subsystems of agents.

Related Work

A number of other agent-oriented software engineering methodologies have tool support including Tropos and O-MaSE (see (Luck and Padgham 2008)) Tropos tool support, consists of a number of separate tools that cover different aspects of the software engineering process. The closest tool to PDT is TAOM4E[3]. The O-MaSE methodology is supported by agentTool[4]. At the 8th Agent Oriented Software Engineering Workshop, held in May 2007, the developers of Tropos, agentTool and PDT presented their tools by demonstrating their use on a popular multi-agent system design case study: a Conference Management System. Papers describing this can be found in the workshop proceedings (Luck and Padgham 2008). The JACK Design Environment (JDE) [5] is also related to our work, since it provides design diagrams. However, the JDE does not support system specification activities or high level design. In addition PDT has some related tools, one of which supports debugging of agent programs based on design models (Padgham, Winikoff, and Poutakidis 2005) and one which supports modifications by application domain experts which can produce fully functioning code (Jayatilleke, Padgham, and Winikoff 2005)

Conclusions and Future Work

Prometheus and PDT try to strike a balance between structured well defined models, and flexibility that is required by engineers. Ongoing work includes increasing the range of automated testing, work on support for maintenance activities, integration with models of social organisation and tools for defining these (Carles Sierra and Winikoff 2007).

We would like to acknowledge the support of Agent Oriented Software Pty. Ltd. and of the Australian Research Council (ARC) under grant LP0453486.

References

Carles Sierra, John Thangarajah, L. P., and Winikoff, M. 2007. Designing institutional multi-agent systems. In Padgham, L., and Zambonelli, F., eds., *AOSE 2006*, 84–103. Springer-Verlag, LNCS.

Jayatilleke, G. B.; Padgham, L.; and Winikoff, M. 2005. A model driven component-based development framework for agents. *Computer Systems Science & Engineering* 4(20).

Luck, M., and Padgham, L., eds. 2008. *Agent Oriented Software Engineering VIII (AOSE'07)*, volume 4951 of *LNCS*. Springer.

Padgham, L., and Winikoff, M. 2004. *Developing Intelligent Agent Systems: A practical guide*. Wiley Series in Agent Technology. John Wiley and Sons.

Padgham, L.; Thangarajah, J.; and Winikoff, M. 2005. Tool support for agent development using the Prometheus methodology. In *ISEAT 2005*. IEEE.

Padgham, L.; Winikoff, M.; and Poutakidis, D. 2005. Adding debugging support to the prometheus methodology. *Journal of Engineering Applications in Artificial Intelligence* 18/2.

Zhang, Z.; Thangarajah, J.; and Padgham, L. 2007. Automated unit testing for agent systems. In *ENASE-07*, 10–18.

[3] http://sra.itc.it/tools/taom4e/

[4] http://agenttool.cis.ksu.edu/

[5] http://www.aosgrp.com/products/jack/index.html