Eclipse-based Prometheus Design Tool [*]

Hongyuan Sun John Thangarajah Lin Padgham
School of Computer Science and IT
RMIT University
Melbourne, Australia
pdt@cs.rmit.edu.au

ABSTRACT

The *Prometheus Design Tool* (PDT) is a graphical tool that is used to design a Multi-Agent System following the *Prometheus Methodology*. This paper describes the latest version of PDT which is now integrated into the Eclipse platform, enabling the users to accomplish the full development life-cycle of an agent-oriented application in one IDE and also inherit the rich set of product development features that Eclipse provides. This version of PDT also aims to support simpler integration with tools from other AOSE methodologies where appropriate.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques

Keywords

Agent Software Engineering; Agent development techniques, tools and environments; Development environments

1. INTRODUCTION

Designing the various components of a multi-agent system(MAS) can be a complex task. The *Prometheus Methodology* [4] provides guidance for an iterative design of the MAS in a top-down manner where designers focus on one particular aspect at a time and eventually identify all the elements of the system in a systematic manner.

There are 3 main design stages: *System specification* where the inputs, outputs, the actors, use cases (scenarios) and the goals of the system are identified; *Architectural design* where roles, agents, communication protocols and the overview of the internals of the system are specified; and *detailed design* where each agents internals are detailed to a level that can be readily implemented. The detailed design is at a conceptual level and abstracted away from any particular implementation allowing the systems to be implemented in the platform of choice, though the code generation feature of the tool currently supports only the JACK Intelligent Agents implementation platform.

PDT[3] [1] is developed to support the *Prometheus* design methodology by offering designers an integrated development environment that provides a graphical interface for system design, auto code generation, type safety, entity propagation and others, some of which we will outline below.

2. PDT FEATURES

The new version of PDT is developed as a plugin to the Eclipse [2] framework. The Eclipse IDE has rapidly grown over the last few years as a popular IDE for developing software in particular Java based applications. Some reasons for its popularity include that it is open source, has a highly extensible framework and an extremely active community. Some of the many built-in features Eclipse provides are file management, project management, version controlling, editing, coding and debugging tools. The PDT Eclipse plugin inherits these rich features and thus upgrading the previous stand alone version of PDT in many ways.

In this new version of PDT we also aim to make it easier to incorporate aspects of other AOSE methodologies. The work in [1] showed that there are many similarities between a number of the major approaches to Agent Oriented Software Design. Nevertheless some approaches have particular aspects or features which could be usefully incorporated into other approaches. Our aim is to simplify the process of interfacing other tools, or parts of tools, to PDT. In earlier work [7] we described how Islander [2] could be integrated with PDT providing additional benefit to both methodologies. In our previous work the integration was hard-wired. Our new Eclipse based framework provides a more modular and robust framework for such integration. We are currently collaborating with the Tracy[6] group to map PDT design to the Tracy architecture, enabling integration of these two approaches.

The above is also facilitated by the new format of the design output file, which is still XML based but clearly separates out the design entities from their respective graphical icons and layout. The new PDT produces a cleaner output file that can be easily managed.

We describe below some of the key features of PDT noting enhancements from the earlier version:

Graphical Editors: Each design phase in Prometheus has a number of associated diagrams representing aspects of the design. These can be graphically created and edited in PDT. The new PDT diagram editing has much improved performance (especially noticeable on large projects) due to the new underlying representation and enhanced algorithms.

The entities are also now represented using the unified graphi-

[*]We would like to thank the many people who have worked on PDT over the years.

Cite as: Title, Author(s), *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. XXX-XXX.

[1]From `http://www.cs.rmit.edu.au/agents/pdt/`
[2]From `http://www.eclipse.org`

cal notations, agreed by the developers of several prominent AOSE methodologies in an attempt to bring a greater coherence to the area. This notation, and the rationale for the various choices is described in [5]. This notation can also easily be changed should a new standard emerge.

The layout of entities is also enhanced by features such as auto-arrange, and the ability to bend connection links to avoid them overlapping with entities.

We also create a miniature view as part of the outline for users to navigate a large-scale diagram more conveniently.

Entity Descriptors: the Eclipse tabbed property view is used to view/edit the details of the properties of the various entities. This information is better organized than in the earlier version of PDT by using different tabs to group related attributes. For example, a data entity has a general *properties* tab (for name, description etc.), a *data fields* tab (describing the fields of the data) and an *events posted* tab which describes any events posted when the data is modified.

Auto Propagation: Propagation of information between different aspects of the design, in order to maintain consistency, has always been an important aspect of PDT, and is a crucial support element even for medium sized projects. Primarily, propagation occurs alongside creation or modification of relations. For example, when a user associates an agent with a role, goals achieved by the role should be automatically propagated to that agent. In the new Eclipse-based PDT, propagation rules are defined in GEF Command[3] as part of the link operation. This allows separation of propagation logic from the rest of the system. This is advantageous for the ongoing evolution and maintenance of PDT, as it is not uncommon that as it is used by different groups with slightly different approaches, new ways of doing things emerge. While consistency must always be maintained, there may be multiple ways of achieving this. The new design more easily allows modification to ask the user which option is desired, when such situations are discovered.

AUML Protocols: PDT employs a modified version of Agent UML (AUML[4]) notation to display interaction protocols within the system. These interaction protocols are defined using a textual notation [8] in PDT. The new version provides an upgraded text editor, which supports syntax highlighting and auto-indentation, and helps to examine for syntax errors of the protocol definition. An *AUML sequence diagram* is generated for a protocol if there are no errors in the textual definition.

Code Generation: PDT is able to generate skeleton code using the detailed design descriptions. It also supports iteration between coding and design, retaining code updates while generating new skeleton code for updated design.

Currently the code is generated for the JACK agent-oriented language[5]. An enhancement of the code generation feature in the new version is that users can define the parameters such as the folder to contain the generated code, the package name and a backup folder that is used when code is regenerated. The ability to define the package name in particular is a simple but necessary enhancement.

Drag And Drop Operation: This simple feature in the new PDT allows for greater ease of modification of a design. In particular, in the old PDT if a particular level (agent or capability) became too complex, it was quite tedious to consolidate some related aspects into a new capability - they had to be manually entered individually into the new entity, links re-established, and then deleted from the old. Now the new capability can be created and the related entities simply dragged and dropped. This supports good design practice of ensuring that each level is kept at an appropriate level of granularity for comprehension.

Report Generation: PDT can export an HTML formatted report for the design which comprises the graphical diagrams and textual information. Users can also choose to export individual design diagrams where the resolution/size of the images can be customized.

3. ONGOING WORK

PDT is actively being developed and extended in line with research work and industry or student feedback. The most important current focus is inclusion of automated testing into the toolkit, and we expect that substantial aspects of this will be integrated during the coming year. We are also keen to provide code generation for a range of platforms, and hope to work with interested parties to incorporate this. We are currently working with the Tracy group to explore how PDT can be used to design Tracy systems and generate appropriate code.

There are also many smaller features which we are constantly working to improve, such as better report generation, enhanced consistency checking, language editing features such as syntax highlighting for JACK code, and the ability to create multiple versions of a design in a scrap book, whilst investigating pros and cons of different choices.

4. REFERENCES

[1] S. DeLoach, L. Padgham, A. Perini, A. Susi, and J. Thangarajah. Using three AOSE toolkits to develop a sample design. *International Journal of Agent-Oriented Software Engineering*, 3(4):416–476, 2009.

[2] M. Esteva, D. de la Cruz, and C. Sierra. Islander: an electronic institutions editor. In *AAMAS 2002*, pages 1045–1052. ACM, July 2002.

[3] L. Padgham, J. Thangarajah, and M. Winikoff. Prometheus design tool. In *AAAI*, pages 1882–1883, 2008.

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

[5] L. Padgham, M. Winikoff, D. Scott, and C. Massimo. A unified graphical notation for aose. In *Agent-Oriented Software Engineering IX: 9th International Workshop*, pages 116–130, May 12-13 2008.

[6] B. Peter, M. Ingo, S. Tino, K. Steffen, W. Volkmar, Schau, and Rossak. Whitestein series in software agent technologies and autonomic computing. In *Tracy: An Extensible Plugin-Oriented Software Architecture for Mobile Agent Toolkits*, pages 357–381. Birkhäuser Basel, 2006.

[7] C. Sierra, J. Thangarajah, L. Padgham, and M. Winikoff. Designing institutional multi-agent systems. In *Agent-Oriented Software Engineering VII, 7th International Workshop, AOSE 2006, Hakodate, Japan, May 8, 2006, Revised and Invited Papers*, volume 4405 of *Lecture Notes in Computer Science*, pages 84–103. Springer, 2006.

[8] M. Winikoff. Defining syntax and providing tool support for agent uml using a textual notation. *Int. J. Agent-Oriented Softw. Eng.*, 1(2):123–144, 2007.

[3] `http://www.eclipse.org/gef/`

[4] `http://www.auml.org`

[5] `www.agent-software.com`