Prometheus: A Methodology for Developing Intelligent Agents

Lin Padgham
linpa@cs.rmit.edu.au

Michael Winikoff
winikoff@cs.rmit.edu.au

RMIT University
Melbourne, Australia

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.2 [**Software Engineering**]: Requirements/Specifications—*Methodologies*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence— *Intelligent agents, Multiagent systems*

1. INTRODUCTION

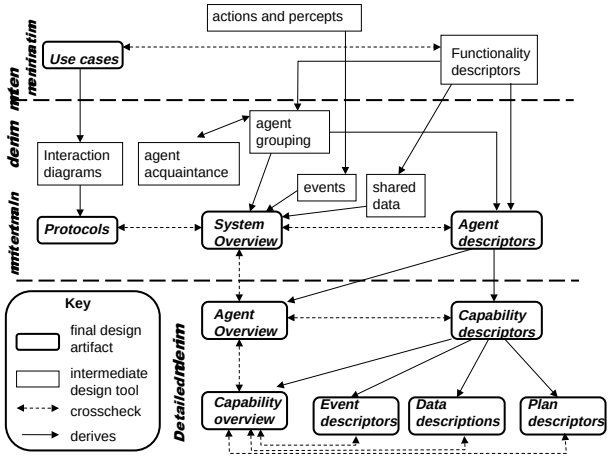
As agents are gaining acceptance as a technology, there is a growing need for practical methods for developing agent applications. This paper presents the *Prometheus* methodology for developing intelligent agent systems. The methodology has been developed over the last several years in collaboration with Agent Oriented Software. Our goal was to develop a design process with associated deliverables which can be taught to industry practitioners and undergraduate students who do not have a background in agents and which they can use to develop intelligent agent systems. Our claim is that Prometheus is developed in sufficient detail to be used by a non-expert. Our evidence is, at this stage, still anecdotal; however, the indications are that Prometheus is usable by non-experts and that they find it useful.

Prometheus has been taught to an undergraduate class of (third year) students who successfully designed and implemented agent systems using JACK [1]. A second year student over the summer vacation was given a description of the methodology and a description of an agent application (in the area of Holonic Manufacturing). With only (intentionally) limited support, the student was able to design and implement an agent system to perform Holonic Manufacturing using a simulator of a manufacturing cell. With student projects it is abundantly clear that the existence of the methodology is an enormous help in thinking about and deciding on the design issues, as well as conveying the design decisions.

Unfortunately space limitations preclude a detailed comparison with the many existing methodologies. We simply note that Prometheu differs from existing methodologies (e.g. [2, 3, 4, 5, 6, 7, 9]) in that it focuses on the development of *intelligent agents* rather than black boxes, supports software engineering activities from requirements specification through to detailed design and implementation, provides detailed processes (not just artifacts and notations), has evolved out of practical industrial and pedagogical experience, has been used by people other than the developers of the methodology, and supports (automatable) cross checking, hierarchical structuring mechanisms and an iterative process.

Although none of these properties is unique in isolation, their combination is, to the best of our knowledge, unique. We believe that these properties are all essential for a *practical* methodology that is usable by non-experts to build real systems, and accordingly the design of Prometheus was guided by these properties.

The *Prometheus* methodology consists of three phases (see diagram below). The *system specification phase* focuses on identifying the basic functionalities of the system, along with inputs (percepts), outputs (actions) and any important shared data sources. The *architectural design phase* uses the outputs from the previous phase to determine which agents the system will contain and how they will interact. The *detailed design phase* looks at the internals of each agent and how it will accomplish its tasks within the overall system.



Instead of attempting to cover all of the activities and artifacts of the methodology we shall focus on the part of the architectural design phase where the agents in the system are identified.

2. IDENTIFYING AGENTS

A major decision to be made during the architectural design is which agents should exist. In the preceding phase (system specification) the *functionalities*[1] of the system were identified. An agent is viewed as a combination of functionalities and so we determine

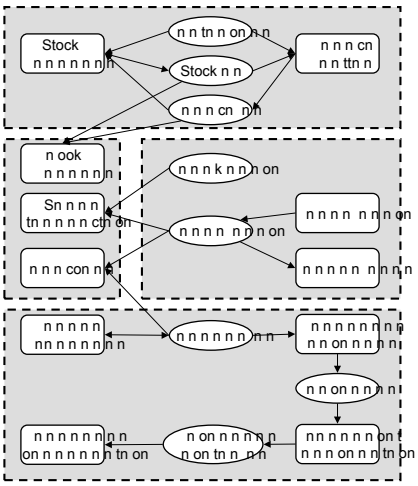
[1]Functionalities (called roles in some methodologies) are things

AAMAS'02, July 15-19, 2002, Bologna, Italy.

the agents that will exist by looking at combinations of functionalities. These combinations are evaluated according to the traditional software engineering criteria of coherence and coupling.

We assess coupling and derive possible groupings using a *data coupling diagram*. The data coupling diagram (see example below[2]) shows functionalities (rounded boxes), data (ovals), and links between them. An arrow from a functionality to data indicates that the functionality produces the data[3]. The diagram assists in identifying which functionalities interact and which functionalities share data; this is used to suggest possible groupings of functionalities into agents. We also identify other reasons for and against grouping functionalities together. For example, if functionalities use the same data it is an indication for grouping them, as is significant interaction between them. Reasons against groupings may be clearly unrelated functionality or existence on different hardware platforms.

In the diagram below we can see that the stock manager functionality and the price setter functionality seem to be strongly coupled and thus we group them together into a single agent type. On the other hand, although the book finder functionality uses both the stock and price databases, it is more related to the user interface functionalities (welcomer and sales transaction) and thus is grouped with them. We do not group the user information functionality with these because we want to have a sales assistant agent (with the welcomer, sales transaction, and book finder functionalities) for each active user, but only a single agent in the system that stores user and bank information[4].



In order to evaluate a potential grouping for coupling we use an *agent acquaintance diagram*. This diagram simply links each agent with each other agent with which it interacts. A design with fewer linkages is less highly coupled and therefore preferable.

3. DISCUSSION AND CONCLUSIONS

We have briefly described one step of the Prometheus methodology. The methodology has been in use for several years and has been taught in industry workshops (most recently at the Australian AI conference, 2001) and to students at RMIT University. The feedback we have received indicates that it provides substantial guidance for the process of developing the design and for communicating the design within a work group.

One of the advantages of this methodology is the number of places where automated tools can be used for consistency checking across the various artifacts of the design process. For example, the input and output events for an agent must be the same on the system overview diagram and on the agent overview diagram. Agent Oriented Software Pty. Ltd. has constructed a support tool for the methodology that allows design diagrams to be drawn and generates corresponding skeleton code in JACK.

Future work includes clearer integration of goals, extension to social agent concepts (e.g. teams, roles), and investigating the use of design artifacts in debugging [8]. We are also in the process of writing a book on the design of intelligent agent systems. This will be a practical "how-to design agent systems" aimed at industrial practitioners and undergraduate students.

Acknowledgements: We would like to acknowledge the support of Agent Oriented Software Pty. Ltd. and of the Australian Research Council (ARC) under grant CO0106934. We would also like to thank James Harland and Jamie Curmi for comments on drafts of this paper.

4. REFERENCES

[1] 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.

[2] S. A. DeLoach, M. F. Wood, and C. H. Sparkman. Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258, 2001.

[3] C. Iglesias, M. Garijo, and J. González. A survey of agent-oriented methodologies. In J. Müller, M. P. Singh, and A. S. Rao, editors, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555, pages 317–330. Springer-Verlag: Heidelberg, Germany, 1999.

[4] D. Kinny, M. Georgeff, and A. Rao. A methodology and modelling technique for systems of BDI agents. In R. van Hoe, editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, 1996.

[5] J. Lind. A development method for multiagent systems. In *Cybernetics and Systems: Proceedings of the 15th European Meeting on Cybernetics and Systems Research, Symposium "From Agent Theory to Agent Implementation"*, 2000.

[6] J. Mylopoulos, J. Castro, and M. Kolp. Tropos: Toward agent-oriented information systems engineering. In *Second International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS2000)*, June 2000.

[7] J. Odell, H. Parunak, and B. Bauer. Extending UML for agents. In *Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence.*, 2000.

[8] D. Poutakidis, L. Padgham, and M. Winikoff. Debugging multi-agent systems using design artifacts: The case of interaction protocols. *Proceedings of Autonomous Agents and Multi Agent Systems (AAMAS)*, 2002.

[9] M. Wooldridge, N. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3), 2000.

that the system needs to be able to do. They are specified using a natural language description and an interface (actions, percepts, interactions with other functionalities, and data produced/used).

[2]The example is taken from an online book store application

[3]Note that we include as data both transient information passed in messages/events as well as persistent data that is stored.

[4]Furthermore, this single agent will probably (for security reasons) be on a different hardware platform.