VNU Journal of Science, Natural Sciences and Technology 27 (2011) 190-204

Agent Oriented Software Engineering: Why and How

Lin Padgham*, John Thangarajah

School of Computer Science and Information Technology, RMIT University, Melbourne, Australia, GPO Box 2476W, Melbourne, VIC 3001, Australia

Received 9 June 2011

Abstract. This paper introduces the concept of agents, and agent systems, and then motivates why developers may want to use this technology for building complex software systems. It describes a particular approach to Agent Software Engineering, the Prometheus methodology, and the associated Prometheus Design Tool. The paper concludes with a discussion of some of the current trends in Agent Oriented Software Engineering.

1. Introduction

In this paper we discuss agent oriented software engineering, trying to answer the question as to *why* you would use this approach, and *how* you would do it. A natural starting place is then to briefly address the question as to "what are agents?"

Software agents are seen by many as a natural evolution from objects, providing an additional level of abstraction and encapsulation (e.g. [1]). A well accepted definition of an agent is from [2], which in turn is adapted from [3]:

"An *agent* is a computer system that is *situated* in some *environment*, and that is capable of *autonomous action* in this environment in order to meet its design objectives."

Wooldridge distinguishes between an agent and an *intelligent* agent which is further required to be *reactive*, *proactive*, and *social* [2, page 23].

Two basic properties of software agents are that they are **autonomous** and that they are **situated** in an environment. The first property, being *autonomous*, means that agents are independent and make their own decisions. This is one of the properties that distinguishes agents from objects. When we consider a system consisting of a number of agents, then a consequence of the agents being autonomous is that the system tends to be decentralised.

Agent technology tends to be used to build systems where the environment is complex and challenging. In particular, in addition to being complex, environments may be *dynamic* - that is the agent cannot assume that the environment will remain static while it is trying to achieve a goal; they may be *unpredictable* in that it is not possible to fully predict the future states of the

* Corresponding author. Tel.: 61-3-9925-3214.
E-mail: lin.padgham@rmit.edu.au

environment; and they may be *unreliable* in that the actions that an agent can perform may fail for reasons that are beyond an agent's control.

Because agents are situated in dynamic environments, they must be *reactive* to changes in those environments, adjusting their plans according to environmental changes. Agents are also *proactive* in that they pursue their own goals. Many agent platforms provide mechanisms to ensure that the agent's behaviour is robust with respect to failures and changing conditions. When the agent behaviour is programmed in terms of high level concepts such as goals, the execution infrastructure ensures that the agent attempts alternative ways to achieve its goals, if initial methods fail. In order to ensure that agents are *robust* and *flexible* they are typically programmed with a number of plans for achieving a given goal. A key issue is balancing reactiveness and proactiveness. An agent's plans and actions should be influenced by environmental changes, and if the agent does not pay sufficient attention to this it may well waste time trying to do things that are either no longer relevant, or no longer possible. However the agent should maintain a focus on its goals, and the achievement of these, and not simply react to its environment. The *social* requirement on agents means they need to interact with other agents, and these interactions are typically framed in terms of conversation *protocols*: patterns of interaction around a particular (goal-oriented) process.

Key properties of an intelligent agent are then the following:

Situated - exists in an environment

Autonomous - independent, not controlled externally

Reactive - responds (in a timely manner!) to changes in its environment Proactive - persistently pursues goals

Flexible - has multiple ways of achieving goals Robust - recovers from failure

Social - interacts with other agents

Discussion of agents and agent systems often distinguishes between *weak* and *strong agency*. Strong agency requires that the agents are modeled in terms of mental attitudes such as beliefs, goals, intentions, plans, commitments, and so on. Perhaps the best known such model is the BDI (Beliefs, Desires, Intentions) model which has its origins in the philosophical work of Bratman [4], but which now has a solid computational body of work encompassing theory, programming languages and platforms, and applications. In these systems agents typically have a collection of *plans*, where each plan is a prescription of steps to achieve a particular *task* or *goal*, and is triggered by an *event* which may arise from the environment, from another agent or from within the same agent's plans. Typically an agent has multiple plans to handle a particular event, each of which are applicable in different situations. An agent has a set of *beliefs* that represent the agent's knowledge about the state of the world and its own internal state.

2. Why are Agents Useful?

Agents, like any other technology, are not magic. Nor do they solve all problems in developing software systems. However they are an approach to structuring and developing software that offers certain benefits, and that is very well suited to certain types of applications. One important aspect of agent systems is that they are distributed and (relatively) decoupled.

This is advantageous for design and development of large complex systems, as well as for the ongoing evolution of such. Jennings also argues that agents are "well suited for developing complex distributed systems" [5] because of the abstraction they provide and the decomposition of complex "nearly-decomposable" systems.

The popular BDI agent paradigm [6] is also very powerful in terms of the flexibility it can provide, in a very modular and easily extensible manner. Each goal will in general have some number of alternative plans that can be used to achieve it. A high level goal will have abstract plans, where the plan steps are further sub-goals, which themselves will have alternative plans. Choices are then made dynamically as to how to achieve each sub-goal. If we take a single goal, and imagine that we have two different abstract plans for achieving that goal, where each plan consists of four subgoals, and then repeat that structure to a depth of three, we will have 146 short plans (4 subgoals each). However we will have over two million ways to achieve the top level goal!!! Because each plan is short and specific, it is relatively easy to add new plans to provide new ways of achieving particular (sub) goals.

Because of the level of abstraction these systems are also faster and simpler to build than traditional systems (once the initial overhead of a new paradigm is overcome). Benfield et. al. [7] have documented a range of benefits of using agent technology in large scale commercial settings. The four major benefits they outlined were:

1. Speed to market
2. Increased productivity
3. Agility in responding to changing/growing requirements
4. Understandability of design

For many business systems it is critical to get the product out the door fast, to establish a niche, possibly then extending the system later. Benfield claims that a number of projects were awarded to their agent-oriented company, simply because they were able to deliver the system faster.

Benfield [7] reports a study in a large logistics company, where they used function point analysis to compare a number of agent based projects, with non-agent based java projects, in order to determine whether to move towards more agent oriented systems. Analysing six agent applications with a total of 7,356 function points and ten person-years of development time, the average productivity on the agent applications was 2.11 function points per day, whereas the average in the company for all other java based projects was 0.45 function points/day. This equates to a 368% improvement. The size of the agent projects ranged from 304 function points to 3,850 function points, while the level of gain ranged from 273% to 513%. This is obviously quite impressive!

Benfield also notes that the largest agent project examined in this study, started as an application with about 350+ function points. Once the benefits were seen by the customer, the requirements quickly increased in size and complexity, to the 3,850 function point system. The BDI agent model scales easily, and it is straightforward to implement an initial "bare bones" system, and then gradually increase the functionality. It is also the case that the building blocks of agent systems are relatively easily understood by clients and users. Consequently the conceptual modeling of the system that is used for requirements

specification, and discussion with clients, is quite close to the concrete design which is then mapped into code.

Although agent systems are still not widely used in industry, there are a number of very successful applications of agent technology. Because of the need for autonomy, NASA is one of the leading users of agent technology. For example Remote Agent [8], in May 1999, was in control of NASA's Deep Space 1 for two days, over 96,500,000 kilometres from the Earth. Other application areas where software agents can provide benefits include Intelligent Assistants [9], Electronic Commerce [10], Manufacturing [11], and Business process modeling [12, 13]. In fact, almost any complex application can benefit from agent technology, although some application characteristics lead to greater benefit from agents than others. In particular complex applications, in unpredictable, changing environments, are where agent systems are particularly useful.

3. Agent Oriented Software Engineering

Whilst traditional software engineering techniques such as Object Oriented modeling can be used to develop agent systems, more specialised techniques that are tailored for agent systems are becoming increasingly popular for developing such systems. These techniques include methodologies and tools that support the complete software development cycle and are referred to as Agent Oriented Software Engineering (AOSE).

AOSE techniques define abstract models in terms of agent concepts (such as agents, goals, plans, tasks, events and communication protocols, rather than the O-O concepts of classes and methods), though an Agent System design will typically also include O-O aspects. These concepts are considered by many to be a more natural means of modeling complex systems. An agent based system is decomposed into multiple, interacting, autonomous agents that have their own objectives to achieve as well as system level objectives that are jointly achieved. Whilst the models available vary between various AOSE methodologies, at an abstract level they all provide means for this system decomposition, objective specification, and for specifying the interaction between the agents.

There have been many AOSE methodologies proposed over the years (see [14, 15]). We briefly mention some of them below:

The GAIA methodology [16, 17] is (arguably) the earliest methodology to gain recognition. It focuses on identification of roles, and the permissions and responsibilities associated with those roles, as well as the protocols they engage in. The MESSAGE [18] methodology abstracts away from specific agent models and identify generic elements that are expressed as meta-models. The INGENIAS methodology [19] extends this work. SADDE [20] is a methodology tailored for building large scale multi-agents that form societies of interacting agents, such as electronic auctions.

PASSI [21] specifies a process for developing agent based systems using UML notation. O-MaSE (previously MaSE)[22] also adapts object oriented techniques and models to the agent paradigm.

The Tropos methodology [23] adopts a requirements driven approach, building on goal oriented approaches for domain and requirements analysis and adapting their analysis methods to the design of agent-based systems.

The Prometheus methodology [24] provides models for each stage of the design process and detailed techniques for developing these models. The methodology is complete in that it covers all aspects of design in detail. We will introduce the Prometheus methodology in Section 4, illustrating how an agent system may be developed.

For a methodology to be useful a graphical tool that follows the methodology is essential. To this end, Tropos, O-MaSE, INGENIAS and Prometheus are supported by TAOM4E[1], agentTool III[2], IDK[3] and PDT[4], respectively. SEAGENT[5] is also a graphical tool for building multi agent systems that follows a Goal-Oriented approach.

4. The Prometheus Methodology and Design Tool

The Prometheus methodology has been developed within the RMIT agents group, in collaboration with Agent Oriented Software, over a period of more than ten years. It is based on experience with companies building agent systems, and the difficulties they experience with the paradigm, as well as experiences with computer science and software engineering students. An integral part of Prometheus is PDT, the Prometheus Design Tool, which guides and assists a developer in designing and modelling an agent system. PDT also produces skeleton code, based on the design model, and supports iteration between design and coding activities. It also supports automated testing[6] based on the design model.

PDT has three main kinds of modelling entities:

- structural graphical diagrams,
- process descriptions,
- detailed descriptor forms.

One of the features of PDT is the way in which it ensures and maintains consistency between different views of the underlying system model being developed. This is extremely important in a large system, where it is virtually impossible to manually check and maintain such consistency. We describe some of the modelling entities, and then follow this with a description of the overall design process using Prometheus.

[1] http://se.fbk.eu/en/tools/
[2] http://agenttool.cis.ksu.edu/
[3] http://grasia.fdi.ucm.es/ingenias
[4] http://www.cs.rmit.edu.au/agents/pdt/
[5] http://seagent.ege.edu.tr/

[6] Currently only unit testing is developed.

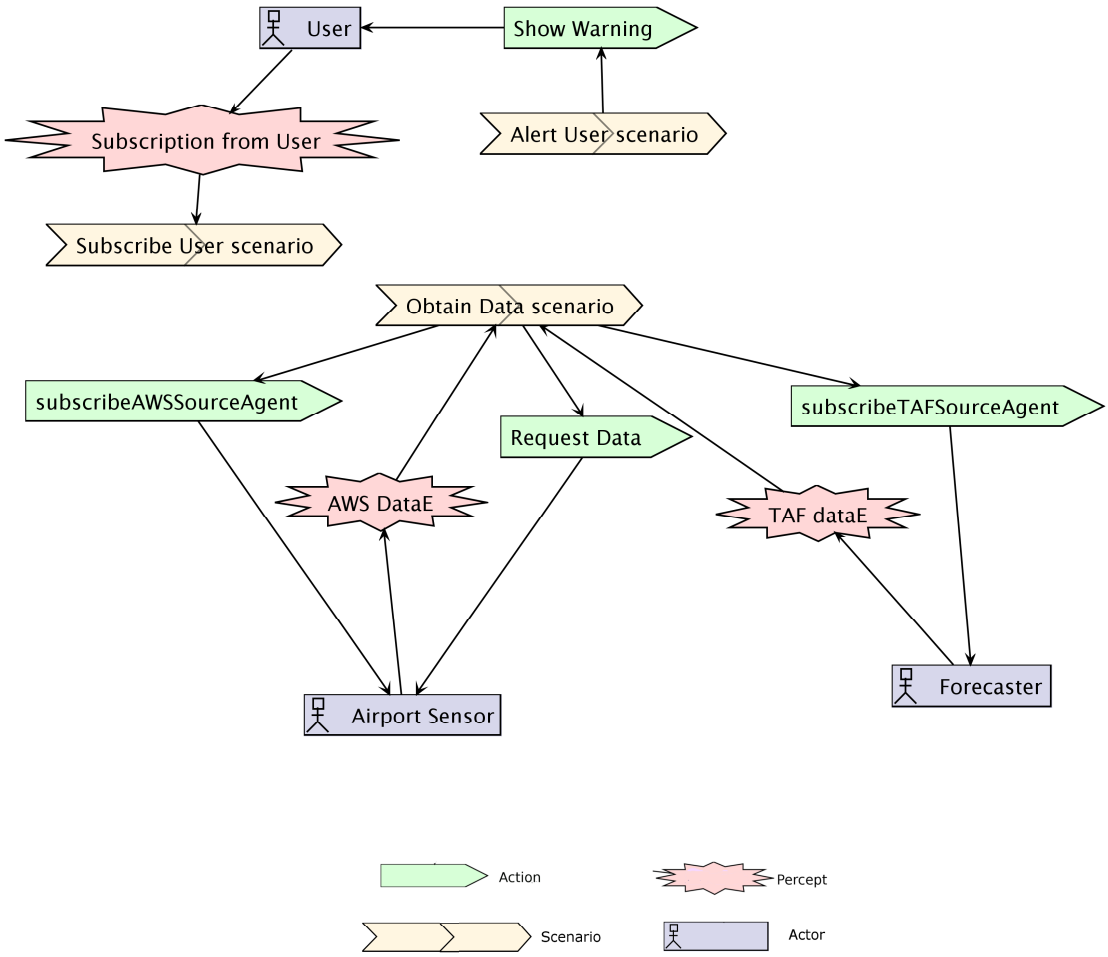

Figure 1. Analysis Overview Diagram.

4.1. Structural Graphical Diagrams

The graphical diagrams of the system structure are the core of a system design done using PDT and Prometheus. These diagrams contain the basic modelling entities of actors, agents, goals, plans, events/messages and protocols. The two initial key diagrams are what we call the "Analysis Overview Diagram" and the "Goal Hiererachy".

An example Analysis Overview diagram from a simplified version of a meteorological warning application for airports, which we built, is shown in figure 1. The function of this diagram is to identify the actors (people or other systems) which will interact with the system under development and the scenarios around which the interaction will happen. We then identify the input to each scenario from the environment or actors, and the output produced by the system from each scenario. Input to the system we call "percepts" and output we call "actions", in line with the standard view of agents as being situated within an environment, receiving percepts and producing actions.

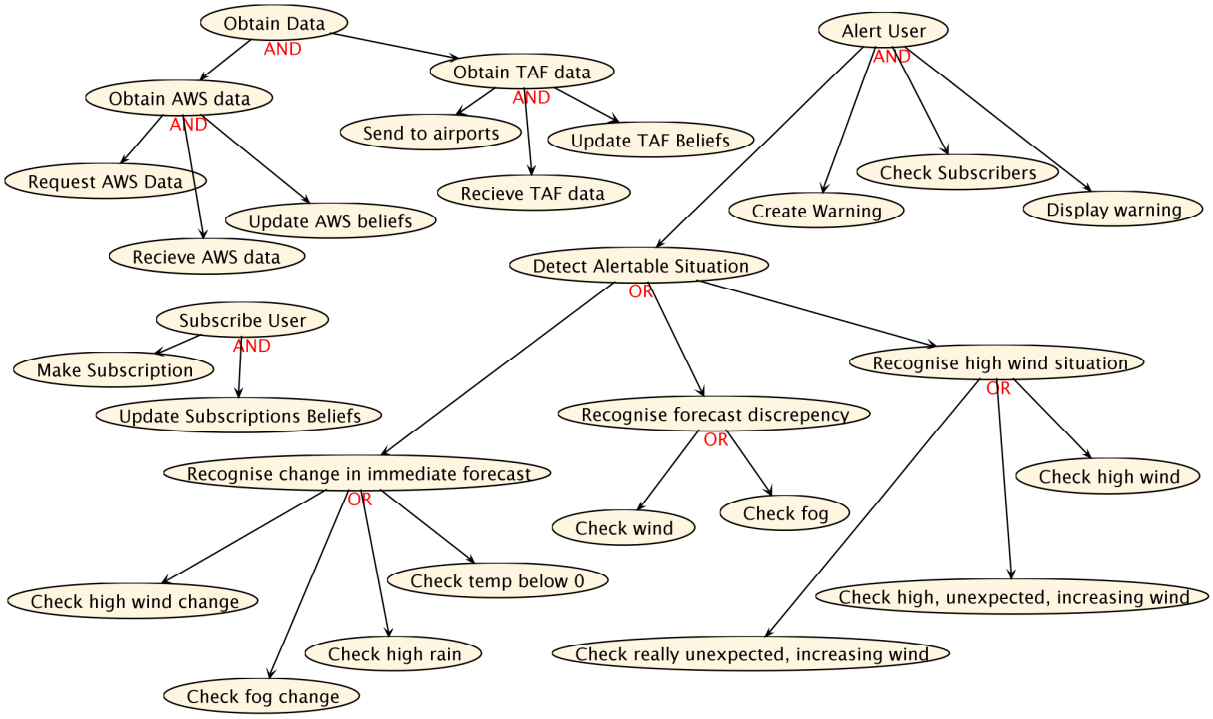

Figure 2. Goal Overview Diagram.

In figure 1 we see three actors: an airport sensor, a forecaster and a user. The two main scenarios are to alert the user (regarding meteorological warnings), and to obtain (meteorological) data. A third scenario is for the user to subscribe to warnings of particular types and at particular locations. We see in this figure also the incoming percepts of data and subscription, as well as the outgoing actions to show a warning, request data, and subscribe to input from the external systems.

An example goal hierarchy diagram is shown in figure 2, where the top level goals of Alert User, Obtain Data and Subscribe User are all propagated from the scenarios specified in the Analysis Overview diagram. These are then broken down into subgoals that are either smaller pieces of the parent goal (AND) or are alternative ways to achieve the parent goal (OR).

The key diagram of the system architecture is the System Overview diagram, showing agents, their interface to the environment via percepts and actions, and their interface to each other via protocols. This diagram will also show any data structures shared between agents, though we usually try to avoid this. In figure 3 we see the system overview diagram for our meteorological application. This diagram is produced almost entirely automatically, from information obtained in various steps of the design process. After developing the Analysis Overview and Goal Hierarchy, goals, percepts and actions are grouped into roles, and these are in turn grouped to form agents. This then provides the information to allow automated placement of

agents and their interfaces into the System Overview diagram. Protocols are defined as part of the process definition and this then allows them also to be automatically inserted and appropriately connected.

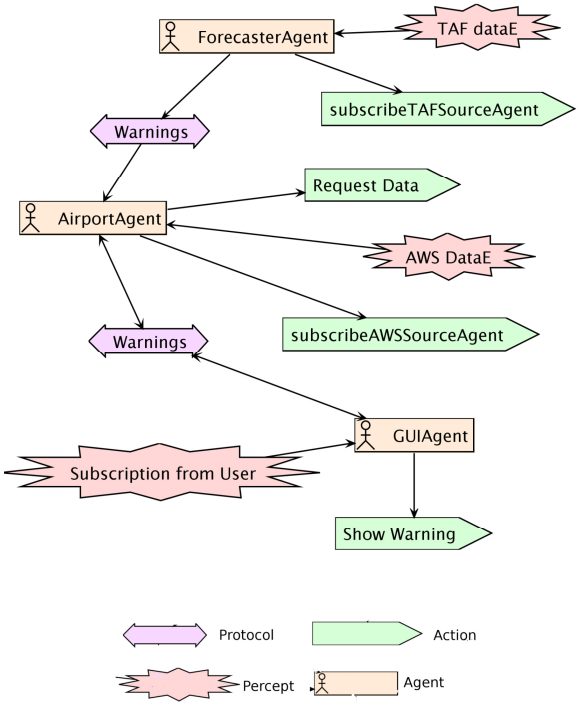

Figure 3. System Overview Diagram.

In this application we see three agent types: a Forecaster Agent which packages and manages forecast data, the Airport Agent which manages information and warnings regarding a particular airport, and a GUI Agent which accepts subscriptions from a user and displays warnings from airports of interest to that user. The agents all communicate with each other within the basic Warnings Protocol of the system, which is shown in figure 6.

The information from the System Overview Diagram is then propagated into the interface of what we call the Agent Overview diagram which has messages (extracted from the protocols), percepts and actions coming into and going out from the agent. The internals of the agent are then shown in terms of plans and capabilities, where capabilities are essentially groupings of plans, messages and data to allow for modularity in design and presentation. The Capability Overview diagram is similar in form to the Agent Overview, and capabilities can be nested. Figures 4 and 5 show an Agent Overview and Capability Overview respectively.

4.2. Process Descriptions

There are two types of process descriptions supported by PDT. These are Scenarios and Protocols. The Prometheus methodology also uses Process diagrams for describing the process undertaken by an individual agent, with respect to a particular task. If the task is a multi agent task, then the internal process will reflect also the incoming and outgoing messages associated with the relevant protocol.

The scenario description outlines a typical way that the scenario might play out, with a focus on percepts, actions and goals as steps in the scenario. A sub-scenario can also be a step. It's purpose is to sketch out how things are exp ected to play out within the system, and to initiate thinking about data, goals, roles, etc. It then also provides a basis for developing protocols.

Protocols utilise the widely used AUML diagrams, that are defined in PDT using a simple textual format but displayed diagrammatically as AUML diagrams, as in figure 6. This shows the main protocol for the meteorology alerting system, which consists of collecting data, and generating warnings. We have extended AUML to show input and output in relation to the protocol (which may involve specific actors). We have found that this increases the understandability and usefulness of the protocol specifications.

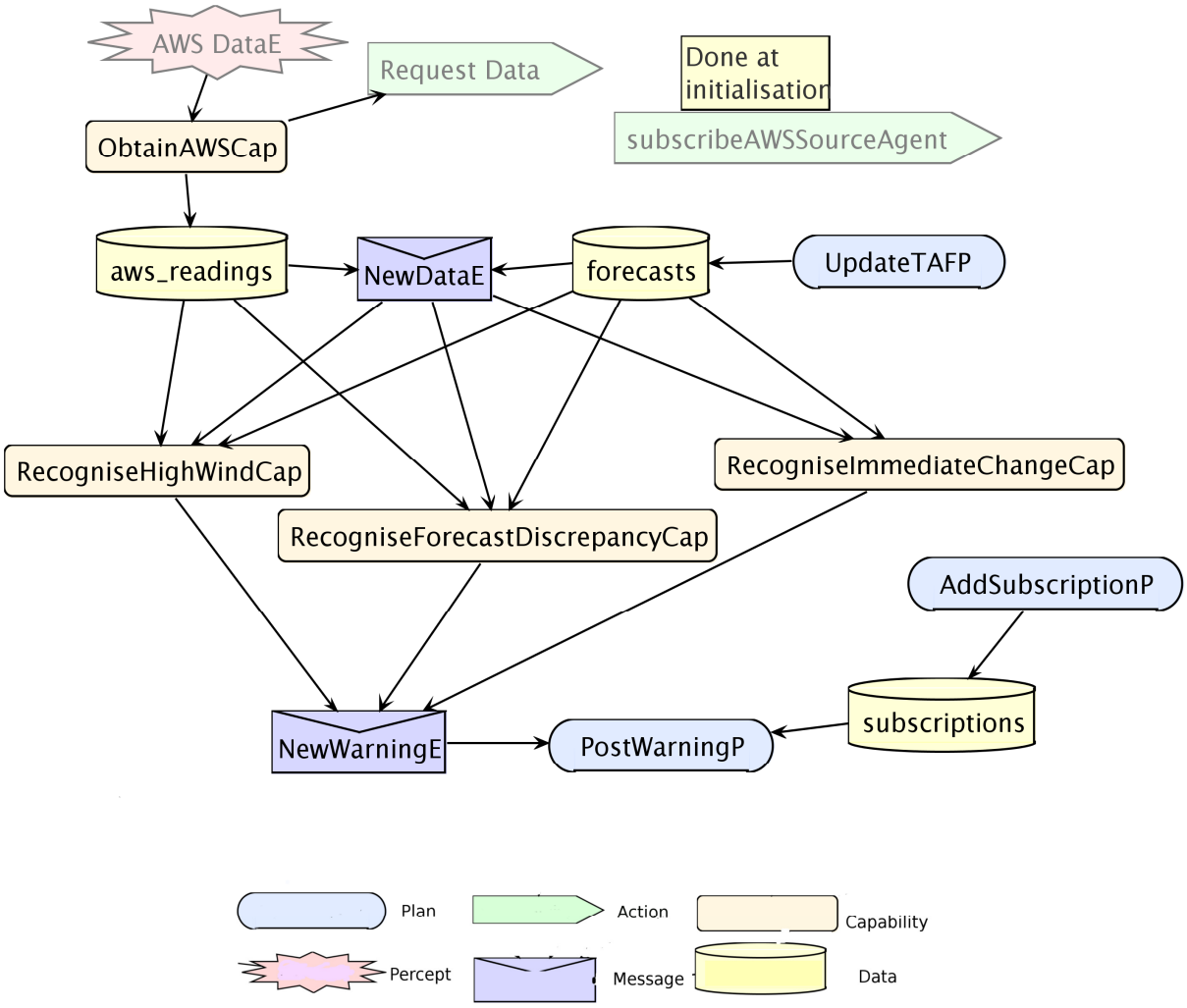

Figure 4. Agent Overview Diagram.

4.3. Detailed Descriptor Forms

The final type of modelling entity in Prometheus and PDT is the Detailed Descriptor Form. These exist for all the types of entity in the system. Much of the information is collected automatically from the structural diagrams, or process specifications, but the descriptor allows all relevant information about an entity to be viewed in the one place. The descriptor form also prompts the designer to consider and document particular design information such as the cardinality of a particular type of agent, initialisation or demise processes for an agent type, whether an event is always expected to have an applicable plan available, and so on.

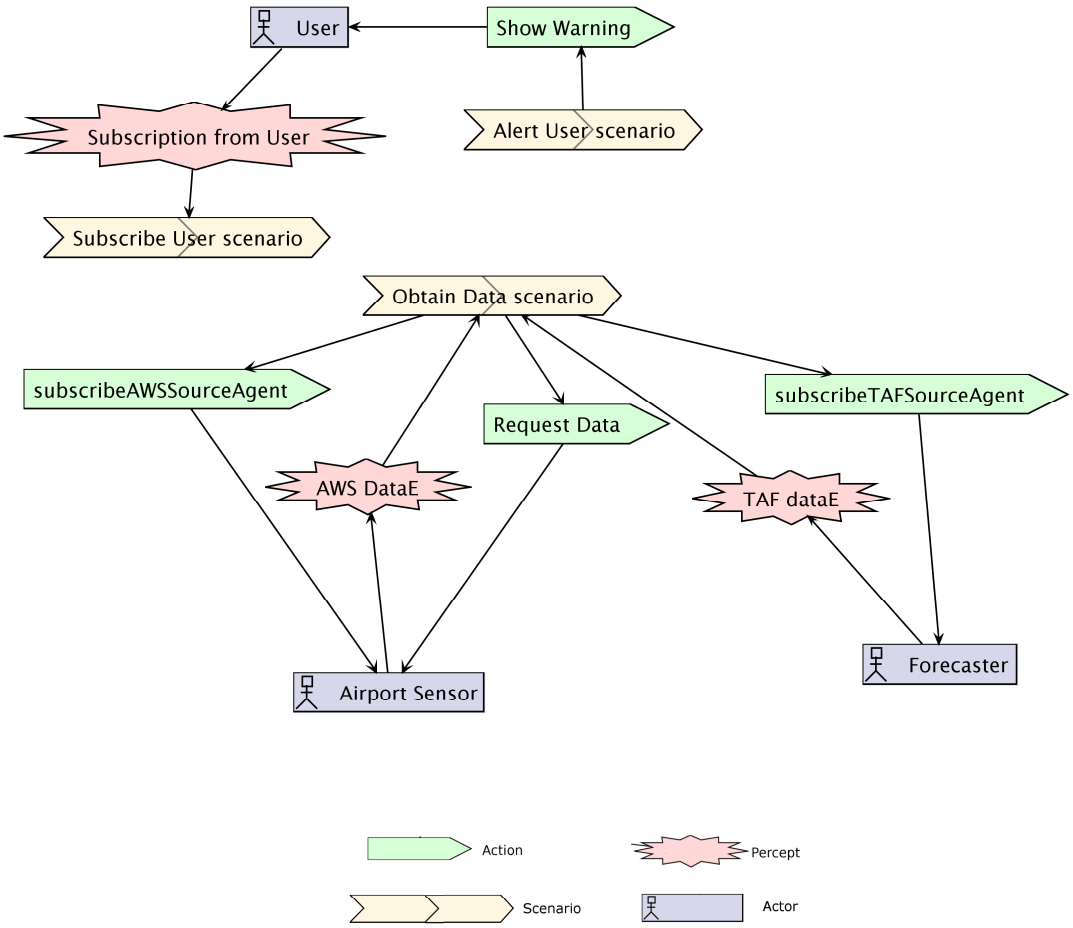

Figure 5. Capability Overview Diagram.

Some of this information is structured (and therefore machine readable, and usable for such things as code generation, testing or debugging) whilst other is just free text and is simply a way of guiding and prompting the developer to consider and to document particular decisions.

4.4. Phases of the Prometheus Methodology

The Prometheus methodology includes the usual phases of system specification, high level or architectural design, detailed design, implementation, debugging, testing and maintenance. These are used iteratively, with more specification and design early on, and more implementation, testing, debugging at later stages. PDT does not currently support integrated debugging or maintenance, although we have research work in both these areas.

In the system specification phase the Analysis Overview and Goal Hierarchy as described earlier are developed. Goals, with percepts and actions are grouped into roles, and necessary data is initially conceptualised. Scenario details are also developed during this phase, usually in parallel with development of the goal hierarchy.

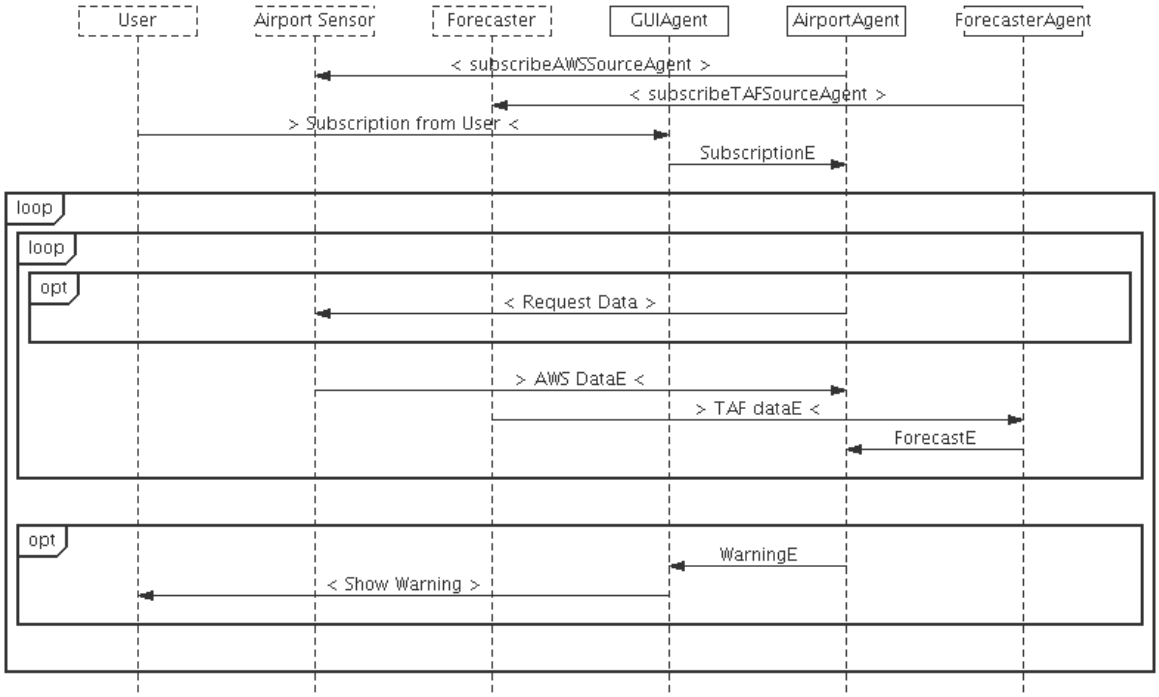

Figure 6. Example Protocol in AUML.

During the architectural design the agent types within the system are identified, based on groupings of the roles identified. Communication between the agents is then specified in terms of protocols, and the individual message types needed at this level are defined.

At detailed design the internals of each of the agent types is developed. Initially this is in terms of capabilities, or modules encapsulating related behaviour. Finally the individual plans are specified, and connected together by subgoal events (or internal messages) to provide the mechanisms by which the agents can achieve the goals they are designed to accomplish within the overall system.

Code generation is based on the models of the architectural and detailed design. PDT supports iteration between design, with automated generation of skeleton code, and manual augmenting of code with details such as plan body computations. Automated unit testing is based on the models of the detailed design phase and requires that skeleton code is generated using PDT.

5. Trends in AOSE

In the past ten years a substantial number of methodologies for designing and building agent systems have been developed (e.g. [14, 15]) and published. A number of these, such as O-MASE [22], Tropos [23], PASSI [21], as well as Prometheus, have quite well developed toolkits. While each of the methodologies has their own strengths and own characteristics, there is actually quite a substantial common core in some of the most well used and well developed

methodologies. This can be seen in the paper by DeLoach et. al. which compares the toolkits for Tropos, O-MaSE and Prometheus and their use in designing a conference management system [25].[7]

As is shown in that paper, although notations and models differ, there is substantial similarity in the concepts used, and even in the particular models developed. There has also been some work done to agree on notation and a paper describing a notation agreed between the developers of PASSI, O-MaSE and Prometheus is described in [27]. At least PDT and AgentTool (the O-MaSE toolkit) are in the process of an upgrade incorporating the new notation.

In general, following a proliferation of methodologies, there is now emerging some substantial consensus on the design entities, and the basic processes associated with designing an agent based system. As the basic design processes of specification, architectural analysis and detailed design are becoming more established, and in many ways converging, more attention is being paid to further aspects such as testing, debugging, maintenance, addition of concepts such as organisations and teams, and integration with Object-Oriented and other standard design environments.

There is also currently a renewed interest in standards for agents. In 1996 FIPA (Foundation for Intelligent Physical Agents) was formed as a body for working on agent standards to allow collaboration between different groups in an open agent environment. Substantial work was done in the late 1990's to produce a standard Agent Communication language (ACL) framework, as well as specifications for agent platforms that would allow heterogeneous agents on different platforms to communicate and collaborate. Following the rise of web services (which are seen by some as a simplified form of agents in an internet environment), there has been a need to revise agent standards to incorporate and complement web service standards. During the early to mid 2000's there was relatively little work on agent standards as the community either used the existing standards or explored the relationships between agents and web services. In 2005 the FIPA standards body voted to become one of the IEEE standards committees, and this body has been gradually growing. This year the Object management Group (OMG) has put out a Request For Proposals (RFP) for Agent and Event standards, and is working on these together with FIPA. This activity may well see new standards emerging in the near future, which may have some impact on the Agent Software Engineering tools becoming available.

At least as important as official standards are defacto standards that emerge from within the community. Eclipse as an IDE is one such defacto standard, and its emergence is reflected in the fact that many of the Agent development tools are now either already integrated into Eclipse, or are in the process of being fully integrated with Eclipse.

6. Conclusion

In this paper we have presented the notion of Agent Systems, and some motivation to use this technology for its power, modularity and efficiency in building complex systems. We have argued that developing agent systems requires a specialised design methodology in order to make effective use of the paradigm. We have described Prometheus and the associated Prometheus Design Tool, PDT,

[7] Similar material though with fewer explicit comparisons can be found in [26]

which is the approach developed within our group over a period of more than ten years, in collaboration with industry specialists, and with much feedback from both students and industry users.

We note that following a period of evolution of a number of agent oriented design methodologies, there now appears to be a period of some emergent convergence and collaboration. At the same time there is a renewed standardisation effort which can be expected to lead to results within the next year or two. As there is convergence on core areas, there is also increasing research into design aspects such as teams and organisations, and additional aspects of the software lifecycle such as testing, maintenance and debugging.

Agent Oriented Software Engineering is a well established sub-field with a specialised journal[8], a longstanding workshop series published by LNCS, and special tracks at both Software Engineering and Agents conferences. It can (perhaps) be expected that over time it will become more and more a standard part of software engineering for complex dynamic systems.

Acknowledgements

There are many people whose input of various kinds must be acknowledged in this work. Firstly we acknowledge the Australian Research Council, who have supported this work under grants CO0106934 and LP0453486, from 2001 to 2007, and Agent Oriented Software who have been our Industry Partner on these grants, and who we have worked with extensively over many years. Secondly we acknowledge Michael Winikoff, who has been a constant part of the RMIT AOSE team from 2001 until 2008. Most of the members of the RMIT Agents group, postgraduate students of this group, and many undergraduate or Masters coursework students have contributed in various ways to Prometheus and PDT. A few deserve a particular mention for their work on PDT: Anna Edberg and Christian Andersson implemented the initial version of PDT during a working holiday in Australia, visiting from Linköping, Sweden; Shankar Srikantiah, Ian Mathieson and Jimmy Sun have all contributed substantially to subsequent versions of PDT; Dave Scerri developed the example design and system used in this paper. We are grateful to all the people, named and unnamed who have contributed in many different ways to this work.

[8] http://www.inderscience.com/IJAOSE

References

[1] J.J. Odell, Objects and agents compared, *Journal of Object Technology* 1 (2002) 41-53

[2] M. Wooldridge, *An Introduction to MultiAgent Systems.* John Wiley & Sons (Chichester, England) (2002) ISBN 0 47149691X, http://www.csc.liv.ac.uk/~mjw/pubs/imas/.

[3] M. Wooldridge, N.R. Jennings, Intelligent agents: Theory and practice. *Knowledge Engineering Review* 10 (1995)

[4] M.E. Bratman, *Intentions, Plans, and Practical Reason.* Harvard University Press, Cambridge, MA (1987)

[5] N.R. Jennings, An agent-based approach for building complex software systems. *Communications of the ACM* 44 (2001) 35-41

[6] M.E. Bratman, D.J. Israel, M.E. Pollack, Plans and resource-bounded practical reasoning. *Computational Intelligence* 4 (1988) 349-355.

[7] S.S. Benfield, J. Hendrickson, J., Galanti, D.: Making a strong business case for multiagent technology. In: AAMAS '06: *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems,* New York, NY, USA, ACM (2006) 10-15

[8] N. Muscettola, P.P. Nayak, B. Pell, B. Williams, Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence* 103 (1998) 5-48

[9] P. Maes, Agents that reduce work and information overload, *Communications of the ACM* 37 (1994) 31-40

[10] M. Luck, P. McBurney, C. Preist,: *Agent Technology:* Enabling Next Generation Computing (A Roadmap for Agent Based Computing). AgentLink (2003) ISBN 0854 327886

[11] W. Shen, D. Norrie,: Agent-based systems for intelligent manufacturing: A state-of-the-art survey. Knowledge and Information Systems, an *International Journal* 1 (1999) 129-156 Extended version available online at http://imsg.enme.ucalgary.ca/publication/abm.htm

[12] N.R. Jennings, P. Faratin, T.J. Norman, P. O'Brien, B. Odgers, Autonomous agents for business process management. International *Journal of Applied Artificial Intelligence* 14 (2000) 145-189

[13] N.R. Jennings, P. Faratin, T.J. Norman, P. O'Brien, B. Odgers, J.L. Alty, Implementing a business process management system using ADEPT: A real-world case study. *International Journal of Applied Artificial Intelligence* 14 (2000) 421- 465

[14] C. Iglesias, M. Garijo, J. Gonz ´alez, A survey of agent-oriented methodologies. In M ¨uller, J., Singh, M.P., Rao, A.S., eds.: *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages* (ATAL-98), Springer-Verlag: Heidelberg, Germany (1999) 317-330

[15] F. Bergenti, M.P. Gleizes, F. Zambonelli, eds.: *Methodologies and Software Engineering for Agent Systems.* Kluwer Academic Publishing (New York) (2004)

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

[17] F. Zambonelli, N. Jennings, M. Wooldridge, Developing multiagent systems: the gaia methodology. *ACM Transactions on Software Engineering and Methodology* 12 (2003)

[18] W. Coulier, F. Garijo, J. Gomez, J. Pavon, P. Kearney, P. Massonet,: MESSAGE: *a methodology for the development of agent-based applications.* [15] chapter 9

[19] J. Pavón, Jorge: Agent oriented software engineering with INGENIAS. In Marik, V., M ¨uller, J., Pechoucek, M., eds.: *Multi-Agent Systems and Applications III,* volume 2691 of LNCS, Springer Verlag (2003) 394-403

[20] C. Sierra, J. Sabater, J. Augusti, P. Garcia,: SADDE: *Social agents design driven by equations.* [15] chapter 10

[21] M. Cossentino, C. Potts, A CASE tool supported methodology for the design of multi-agent systems. In: *Proceedings of the International Conference on Software Engineering Research and Practice* (SERP'02), Las Vegas (20 02) Available from http://mozart.csai.unipa.it/passi/.

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

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

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

[25] S.A. DeLoach, L. Padgham, A. Perini, A. Susi, J. Thangarajah, Using Three AOSE Toolkits to Develop a Sample Design. The International *Journal of Agent Oriented Software Engineering* (2009) To appear

[26] M. Luck, L. Padgham, eds.: Agent Oriented Software Engineering VIII (AOSE'07). *Springer,* LNCS (2008)

[27] L. Padgham, M. Winikoff, S. DeLoach, Cossentino, M. LNCS. In: *A Unified Graphical Notation for AOSE. Springer-Verlag* (2009)

Công nghệ phần mềm hướng tác tử: Vì sao và làm thế nào?

Lin Padgham, John Thangarajah

Đại học Công nghệ Hoàng gia Melbourne, Australia,
GPO Box 2476W, Melbourne, VIC 3001, Australia

Bài báo này giới thiệu các khái niệm về tác tử và các hệ tác tử, và sau đó đưa ra những thuyết minh nhằm thúc đẩy việc sử dụng công nghệ này để xây dựng những hệ thống phần mềm phức tạp. Bài báo mô tả một cách tiếp cận cụ thể đối với Công nghệ phần mềm hướng tác tử - phương pháp Prometheus, và công cụ thiết kế Prometheus đi kèm. Bài báo cũng thảo luận về một số hướng nghiên cứu hiện tại trong Công nghệ phần mềm hướng tác tử.