

Agent-based Software Development Methodologies

Brian Henderson-Sellers, University of Technology, Sydney (Australia)
and Ian Gorton, Pacific Northwest National Laboratory (USA)

Abstract

In this White Paper, produced as a result of discussions at the OOPSLA 2002 Workshop on Agent-Oriented Methodologies, we outline the current state of play of agent-oriented methodologies, how they might be integrated into an underlying, metamodel-based framework, and what the research community needs to do to make their products acceptable to industry. We conclude with an invitation to the community.

Introduction

Software agent technologies are an emerging branch of distributed artificial intelligence research. There is widespread and growing interest in the potential of software agent technologies for building the next generation of advanced software systems. This interest is especially focused in domains such Internet-based applications and problems requiring advanced user interactions and flexible collaborations.

The term software agent is gaining considerable attention in the software engineering and information technology community. So-called intelligent, or autonomous, agents are reportedly capable of many advanced forms of behavior. Advocates of agents claim many benefits from their use. These include better ways to structure applications, improvements in programming language abstractions for complex inter-agent communications, and a natural metaphor for human-computer interaction (Jennings *et al.*, 1998).

Software agents and related technology have implications for software engineering. As autonomous software entities, agents represent potential components in a software system. The advanced agent communication mechanisms further represent ways to structure societies of agents that need to exchange information and cooperate. The increasing numbers of technologies, including some COTS products that directly support agent abstractions, are implementation alternatives for an application. Agents are therefore something a software engineer must begin to consider as the technology transitions from the research labs to products.

What is an Agent?

Agent technologies have emerged from the field of distributed artificial intelligence research (Jennings *et al.*, 1998). The agent community itself has many definitions of what exactly constitutes an agent system. Franklin and Graesser (1997) point out that many of

these are broad and encompass a wide range of software systems, many of which were not conceived or designed as agent systems.

As an example of widely cited definitions of agents and their capabilities, Jennings *et al.* (1998) defines software agents as having the following characteristics:

- They are situated in some environment
- They are capable of flexible autonomous action in order to meet design objectives

The notion of flexibility is further expanded to incorporate:

- Responsiveness: agents perceive their environment and respond in a timely fashion to changes that occur in it
- Pro-active: agents should exhibit opportunistic goal-driven behavior
- Social: agents should be able to interact in order to complete their own problems and help others with their activities

In addition, Wooldridge and Jennings (1999) state that software agents are said to be appropriate for applications in which:

- Data, control, expertise or resources are distributed
- Agents provide a natural metaphor for delivering system functionality
- A number of legacy systems must be made to interwork

Abstraction Issues Explored

The key role of abstraction is well known in the advancement of software engineering. Abstraction has been responsible for the evolution of programming languages through the adoption of higher level mechanisms for code construction. Abstract mechanisms are introduced in to languages to replace and enforce useful constructs that were only previously possible through hand-coding compositions of multiple low-level operations. Very simplistically, using better abstractions means you have to write less code and make less errors, and can therefore tackle more complex problems in the same time-scale¹. This pattern has been repeated many times in the software community, for example:

- COBOL, a high level language was introduced to provide an English-like abstraction for describing program operations that at the time were usually written in terms of obscure assembly language operations.
- Object-oriented languages introduced programming constructs for data encapsulation in modules, type extension through inheritance, and polymorphic substitution mechanisms for sub-types that supported improved structuring and organization of program modules over procedural techniques.
- Distributed technologies such as CORBA provide abstract mechanisms that greatly simplify the construction of type-safe inter-process communications, as compared to using raw protocols or sockets.

¹ Abstraction has many other benefits, such as eliminating errors, providing higher level building blocks for design and construction, and so on.

Given this evolution pattern for software technology, it is hardly surprising that the agent technologies introduce new abstraction mechanisms for application construction (Wooldridge *et al.*, 2000). Many technologies for constructing agent applications provide extensions to standard programming languages such as Java in order to facilitate advanced inter-agent communications mechanisms and directly implement agent management policies and semantics. These new programming language abstractions are the most concrete contribution of agent technology to software construction.

Agent-based Software Development Methods

There is an increasing amount of work in extending agent concepts to encompass broader software engineering lifecycle activities (e.g. Griss and Pour, 2001; Tveit 2001) such as design and decomposition, but this is in a nascent stage. These emerging methods attempt to exploit the key ideas behind agents at various stages of the software development lifecycle. Typically, the currently published methods have an emphasis on some particular part of the software development process, for example Tropos (e.g. Mylopoulos *et al.*, 2001; Bresciani *et al.*, 2003) emphasizes early requirements. There is a spectrum of approaches and techniques promoted by these methods. However, an underlying theme of many (but not all – see further discussion in next section) is the extension or customization of many of the object-oriented software development concepts that have been adopted over the last decade (e.g. Wooldridge *et al.*, 2000). Figure 1 depicts a genealogy of many of the current agent-oriented methodologies. This is not surprising given the reliance of agent programming abstractions upon object-oriented languages.

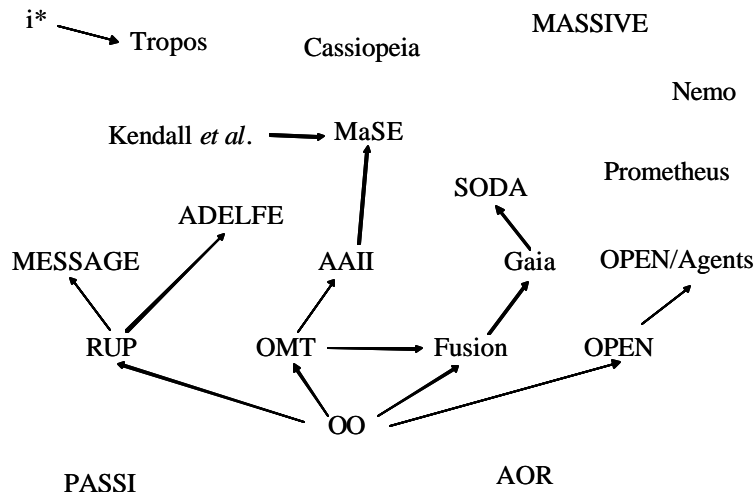


Figure 1 A genealogy of some of the current agent-oriented methodologies

Maturing Agent-based Development Methods

Whilst multi-agent² systems have yet to impact the commercial world significantly (i.e. there are very few agent-oriented commercial systems), multi-agent technology offers

² A single agent is of little interest; rather, commercial systems built using agent-oriented software engineering will be multi-agent systems. The interactions between autonomous components are the essence of multi-agent technology — the source of emergent behaviour.

substantial promise as the foundation in the near future for commercial systems distributed across the new economy (Jennings *et al.*, 1998). In particular, multi-agent technology may provide a potent solution to the problems of scalability, interoperability and flexibility through the use of autonomous components. The deployment of multi-agent systems has been held back by the lack of commercially acceptable design methodologies and by their high cost. Agent-oriented software engineering is a very active area of research (see Jennings and Wooldridge, 2001 and papers in Debenham *et al.*, 2002) and a wealth of interesting ideas have been published in the past two years (e.g. Mylopoulos *et al.* (2001), Bresciani and Giorgini (2002) and Bresciani *et al.* (2003) on Tropos; and Padgham and Winikoff (2002) on Prometheus.)

There are essentially two schools of thought. One dictates that agent-oriented methodologies should be developed independently of object-oriented methodologies. Tropos is one such example. The rationale is that in the construction of the methodology, an OO methodology uses concepts of classes, class features and a specific set of relationships based on the object client-server model. The alternative approach is to take an existing OO methodology and extend it to support agent concepts. Such an approach is taken by Wooldridge *et al.* (2000) whose Gaia AO methodology sits on top of ideas from the OO methodology of Fusion (Coleman *et al.*, 1994). A second example is seen in the proposals to extend the OO methodology OPEN (Graham *et al.*, 1977) to support agent technology (Debenham and Henderson-Sellers, 2002, 2003).

One proposal of this White Paper, reflecting the discussions at the OOPSLA 2002 Workshop on Agent-Oriented Methodologies, is to use an integrating metamodel-based framework such as the OPEN Process Framework (OPF) to “host” the merger of ideas from AO methodologies – particularly useful for those that are truly only fragments of methodologies. We would also like to facilitate the convergence, via consolidation and method engineering, of a wide range of full AO methodologies such as Tropos, Gaia, MaSE, Prometheus, AOR and Passi, including linking to the FIPA group with this focus.

In the next section, we outline the OPEN Process Framework so that readers can evaluate the usefulness of process engineering using an existing metamodel-based framework to approach the consolidation challenge outline above.

A starting point: the OPEN Process Framework

We now briefly describe a suitable, existing process infrastructure upon which to build a facility to support the design of multi-agent systems. OO methodologies that are highly prescriptive and overly specified are hard to extend when a new variant or a new paradigm appears. What is required is a more flexible approach to building methodologies or processes. One such will be described here: OPEN (Object-oriented Process, Environment and Notation: Graham *et al.*, 1997; Henderson-Sellers *et al.*, 1998). Using OPEN, process components are selected from a repository and the actual methodology (or process) is constructed using identified construction and tailoring guidelines. OPEN thus provides a useful starting point because it is not only defined at the metamodel level but is also itself componentized. Thus, adding further support for the design of intelligent agents is feasible, such potential extensions having been *a priori* designed into the metamodel architecture of the OPEN Process Framework (OPF).

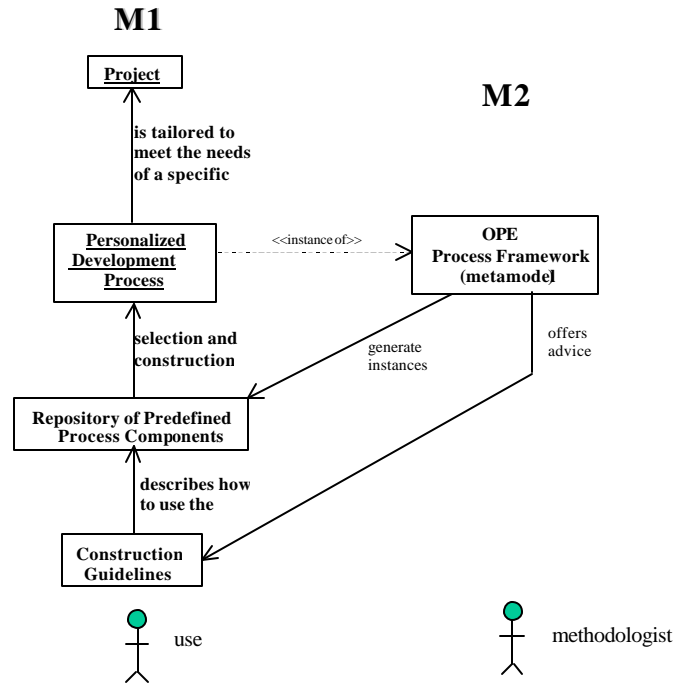


Figure 2 Creating a personalized development process

The OPF is a process metamodel or framework from which can be generated an organizationally-specific process (instance) (Figure 2). Some of the major elements in this metamodel (Figure 3) are Work Units (Activities, Tasks and Techniques wherein Activities and Tasks say “what” is to be done and Techniques say “how” it will be accomplished), Work Products and Producers³. Together, Work Units and Producers create Work Products and the whole process is structured temporally by the use of Stages (phases, cycles etc.). Each process instance is created by choosing specific instances of Activities, Tasks, Techniques etc. from the OPF Repository (Figure 2) and specific configurations thereof (created by the application of the Construction Guidelines). OPEN thus provides a high degree of flexibility to the user organization.

Initial work in identifying new process components (e.g. Activities, Tasks, Techniques, Roles, Work Products) has already started. For example, in a newly proposed Task named *Identify intelligent agents* we suggest that identification of agents is in some ways an extension of ‘finding the objects’, although there are several clear differences (e.g, Odell, 2000). Agents are autonomous entities that have many similarities to objects. A major difference is that whereas an object receiving a request for service must deliver that service, an agent is empowered to say ‘no’. Agents act when “they feel like it”, and not necessarily when they receive a communication or other stimulus. Agents play roles with responsibilities. These responsibilities are not only equivalent to those for objects

³ Interestingly, the Tropos methodology is said to supply the “why” thus making it an ideal complement to the OPF.

(responsibilities for doing, knowing and enforcing) but also towards achieving organizational goals (Jennings, 2001). However, they attempt to more closely mimic the behaviour of people and their decision making strategies than can objects. Consequently, there is a greater emphasis on the *roles* that are played by agents. Each role is defined by four attributes: responsibilities, permissions, motivations and protocols (Wooldridge *et al.*, 2000). Roles are already well supported in OPEN but need to be extended and refined to support the more sophisticated notion of roles in agent technology (e.g. Cabri *et al.*, 2002).

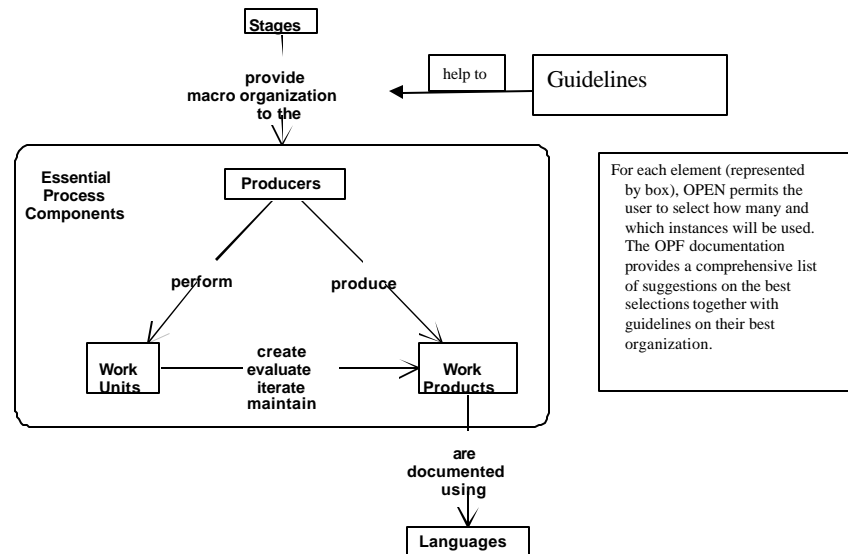


Figure 3 The major elements of the OPF metamodel

On the other hand, the Tropos methodology (e.g. Mylopoulos *et al.*, 2001; Bresciani and Giorgini, 2002) utilizes *agent* (rather than object) concepts in the very structure of the process. At the same time, the underpinning metamodel (e.g. Bresciani *et al.*, 2003) is compatible with and could easily form an extension to the OPF metamodel. Tropos originated in the requirements engineering (for agents) community and discriminates clearly and strongly between early requirements modelling (domain and business focussed) and late requirements engineering (focussing on software requirements and logical design). As such it contains significant knowledge of agent modelling applied to process.

Thus, agent modelling has parallels with the use of roles in object modelling. An overall “community of agents” can therefore be well modelled at the highest level of abstraction using techniques of role modelling, collaboration diagrams and other standard OO structuring techniques and diagrams. Some extensions to the Unified Modeling Language have recently been proposed by Odell *et al.* (2000) in order that that modelling language may be considered applicable for agent modelling. These standard modelling techniques and design notations will therefore not be discussed further here since we will instead focus on the new additions to the developer’s suite of tools and techniques.

Industry Adoption

One of the major differences between industry adoption of agent technology in comparison to its adoption of object technology in the early 1990s (and continuing) is the

lack of specifically agent-oriented programming languages. Although programming languages are only part of the development story, industry is reticent to adopt a new paradigm *at the conceptual level* if it is then impossible to implement these ideas in a currently acceptable, commercially viable programming language. Industry also requires there to be a clearer explanation offered as to the benefits of agent technology. At present they appear reticent to adopt agent technology since no one can show where agents succeed and objects fail.

Another issue which is addressed solidly in this white paper is whether the availability and marketing of multiple methodologies is an obstacle to widespread industry adoption. A consolidated approach to further development in a collaborative rather than a competitive mode would appear to address this concern. Using a combination of best of breed methodologies, perhaps using process engineering as discussed here, might give a clearer signal to industry and a reassurance of some stability in the AO methodology “product”.

An Invitation

To incorporate ideas from currently available AO methodologies into existing methodological frameworks such as the OPF (Firesmith and Henderson-Sellers, 2002) is a major challenge which the AO methodologist community needs to address. This challenge and opportunity identified at the OOPSLA 2002 workshop has been independently identified by FIPA who have recently (early 2003) set up a working group to address the identical issue. Liaison between the OOPSLA group and the FIPA group is being established by overlaps in their membership. We invite you to join an increasing collaboration between AO methodology groups worldwide in order to provide both industry and academe with a strong foundation for full lifecycle development of agent-oriented systems. Email brian@it.uts.edu.au.

References

- Bresciani, P. and Giorgini, P. (2002). The TROPOS analysis process as graph transformation system, in Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies (eds. J. Debenham, B. Henderson-Sellers, N. Jennings, and J. Odell), COTAR, Sydney, 1-12
- Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J. and Perini, A., 2003, Tropos: an agent-oriented software development methodology, J. Autonomous and Multi-Agents (in press)
- Cabri, G., Leonardi, L. and Zambonelli, F. (2002). Modeling role-based interactions for agents, in Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies (eds. J. Debenham, B. Henderson-Sellers, N. Jennings, and J. Odell), COTAR, Sydney, 13-20

Debenham, J.K. and Henderson-Sellers, B. (2002). Full lifecycle methodologies for agent-oriented systems – the extended OPEN Process Framework. In Proceedings Agent-Oriented Information Systems (eds. P. Giorgini, Y. Lesperance, G. Wagner and E. Yu), Toronto, 87-101

Debenham, J.K. and Henderson-Sellers, B. (2003). Designing agent-based process systems –extending the OPEN process framework, chapter in Intelligent Agent Software Engineering (eds. V. Plekhanova), Idea Group Publishing, 160-190

Debenham, J., Henderson-Sellers, B., Jennings, N. and Odell, J. (2002). Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies, COTAR, Sydney, 130pp, ISBN 0-9581915-6

Firesmith, D.G. and Henderson-Sellers, B. (2002). The OPEN Process Framework. An Introduction, Addison-Wesley, Harlow, UK, 330pp

Franklin, S. and Graesser, A. (1997). Is it an agent, or just a program?, **in** *Intelligent Agents III, LNAI vol 1193*, Springer-Verlag, Berlin, 21-36

Graham, I., Henderson-Sellers, B. & Younessi, H. (1997). The OPEN Process Specification. Harlow, UK: Addison-Wesley, 314pp.

Griss, M.L. and Pour, G. (2001). Accelerating development with agent components, *IEEE Computer*, **May 2001**, 37-43

Henderson-Sellers, B., Simons, A.J.H. & Younessi, H. (1998). The OPEN Toolbox of Techniques. Harlow, UK: Addison-Wesley, 426pp + CD.

Jennings, N. (2001). Agent of change. *Application Development Advisor*, 5(3), 6.

Jennings, N.R. and Wooldridge, M. (2001). Agent-oriented software engineering. In J. Bradshaw (ed.), *Handbook of agent technology*. Cambridge, MA, USA: AAAI/MIT Press.

Jennings, N.R., Sycara, K. and Wooldridge, M. (1998). A roadmap of agent research and development. *Int. Journal of Autonomous Agents and Multi-Agent Systems*, 1 (1), 7-38.

Mylopoulos, J., Kolp, J. and Castro, J. (2001). UML for agent-oriented software development: the Tropos proposal, in «UML»2001 – The Unified Modeling Language (eds. M. Gogolla and C. Kobryn), LNCS Vol 2185, Springer-Verlag, 422-441.

Odell, J. (2000). Objects and agents: how do they differ? *JOOP*, 13 (6), 50-53.

Odell, J. (2001). Key issues for agent technology, *JOOP*, 13 (9), 23-27, 31.

Odell, J., Van Dyke Parunak, H. and Bauer, B. (2000). Extending UML for agents, in Procs. Agent-Oriented Information Systems Workshop, 17th National Conference on Artificial Intelligence (eds. G. Wagner, Y. Lesperance and E. Yu), Austin, TX, USA, 3-17

Padgham, L. and Winikoff, M. (2002). Prometheus: a pragmatic methodology for engineering intelligent agents, in Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies (eds. J. Debenham, B. Henderson-Sellers, N. Jennings, and J. Odell), COTAR, Sydney, 97-108

Rao, A.S. and Georgeff, M.P. (1995). BDI agents: from theory to practice. In Procs. First International Conference on Multi-Agent Systems. San Francisco, CA, USA, 312-319.

Tveit, A. (2001). A survey of Agent-Oriented Software Engineering. In *Proceedings of the First NTNU Computer Science Graduate Student Conference*. Norwegian University of Science and Technology, May 2001, <http://csgsc.idi.ntnu.no/2001/pages/papers/atveit.pdf>

Wooldridge, M.J. and Jennings, N.R. (1999). Software engineering with agents: pitfalls and pratfalls, *IEEE Internet Computing*, **May/June 1999**, 20-27

Wooldridge, M., Jennings, N.R. and Kinny, D. (2000). The Gaia methodology for agent-oriented analysis and design. *J. Autonomous Agents and Multi-Agent Systems*, 3, 285-312.