The Knowledge Engineering Review, Vol. 12:3, 1997, 323–329

Methodological foundations for agent-based systems*

MICHAEL FISHER[1], JÖRG MÜLLER[2], MICHAEL SCHROEDER[3], GEOF STANIFORD[4] and GERD WAGNER[5]

[1]*Department of Computing, Manchester Metropolitan University, UK*
[2]*Agent Systems Group, Zuno, London, UK*
[3]*Institut für Rechnergestützte Wissensverarbeitung, Universitaät Hannover, Germany*
[4]*Department of Computer Science, University College Chester, UK*
[5]*Institut für Informatik, Universität Leipzig, Germany*

In spite of the rapid spread of agent technology, there is, as yet, little evidence of an *engineering* approach to the development of agent-based systems. In particular, development methods for these systems are relatively rare. One of the key reasons for this is the inadequacy of standard software development approaches for these new, and fundamentally different, agent-based systems. Traditional software development methods often lack the flexibility to handle high-level concepts such as an agent's dynamic control of its own behaviour, its ability to represent cooperative interactions, and its mechanisms for representing internal change, assumptions, objectives, and the uncertainty inherent in its interactions with the real-world.

1 Why principled development?

Perhaps the first question to consider is whether development methodologies are appropriate. Is agent technology advancing so rapidly that no one will ever have the time to undertake principled development?

While the question is relevant whether we are considering agents or any other new technology, the rapidity with which agent-based systems are being developed makes the answer particularly important.

The speed of technological advance is related to how well the technology itself is understood. It is therefore most unlikely that agent technology can continue to move fast (if at all) without principled development. Without it, system engineers will never know for certain what their system does. Once you have well-tried principles you can use them to good advantage (e.g. checking that you have covered all possibilities, analysing complexity issues, identifying limitations to the system you have designed). Testing, reengineering, and reusing software also rely on principled development.

While undoubtedly, as with traditional systems, few people will use agent development methods for small systems, when larger or more complex agent-based systems are developed in the future these methods may come in to their own. At a pragmatic level, when substantial systems are built by large teams of software engineers, the use of (approved) development methods is likely to be mandatory.

* This report is the result of a panel discussion at the *First UK Workshop on Foundations of Multi-Agent Systems* (FoMAS'96). All members of the panel are the authors, listed alphabetically, with the exception of Aaron Sloman and Brian Logan from the University of Birmingham, who were unable to integrate their views (which differ from those expressed here) into this document due to time constraints.

2 Developing agent-based systems

How can we develop agent-based systems, from high-level requirements or specifications through to implementations?

In general, developing agent-based systems has much in common with developing other complex software. Therefore, many of the problems and solutions known from general software engineering apply to agent-based systems as well. In addition, an approach in which the key characteristics of agents (e.g. goal generation and autonomy) are central is required.

How agent-based systems can be developed depends not only on the purpose for which they are developed, but also on whether we can afford the luxury of developing a stand-alone system from scratch, or whether we must provide a subsystem that needs to interact with legacy systems. In the former case, the focus is likely to be on the actual process of agent modelling; in the latter case, it is on component interfaces. Industrial agent-based systems are likely to fall under the latter category; therefore, a large effort is necessary to provide a framework that allows agents to communicate with legacy systems, for example by wrapping these legacy systems and giving them a clean interface that allows agents in the system to interact with them.

Having created a uniform perspective of a heterogeneous system, the next question is what the software development cycle that normally underlies the development of a system should look like in the case of agent-based systems. Given the variety of agent definitions (e.g. Franklin and Grasser, 1997), this is difficult to assess. However, a methodology that embraces both the autonomy and the high-level cooperative aspects of agents seems essential. Currently, no one approach is entirely satisfactory and it may be the case that it is either impossible or undesirable to try to develop one all-embracing method given the rich diversity of approaches under the heading of agent-based systems.

In recent years, there has been a growing tendency to standardize traditional software application development by choosing a relational database as the kernel of an application and using declarative database programming languages for specific tasks. Agent-based systems may be implemented in a similar way: there could be standardized knowledge systems which form the kernel of agent-based systems, together with declarative agent specification languages, possibly extending SQL and including certain object-oriented features, which provide the operations of knowledge-based inference and update needed for intelligent agents. Indeed, there are already emerging standards for particular aspects of agent systems such as the KQML language for agent communication, while other, broader, efforts (specifically by FIPA) are concerned with the promotion of interoperable agent systems through the standardization of several component technologies for agents, including such agent communication languages.

However, as there is no generally accepted and well-defined taxonomy for classifying agent-based applications, it is too early to assess which development approaches will be appropriate for which classes of agent-based systems.

3 Using structured methods?

In developing agent-based systems, can we use variations on traditional structured methods?

It seems likely that, just as in 'standard' software engineering, structured methods will be the most widely used of the (future) agent development approaches. Currently, however, structured methods are not suitable for agent-based systems since they are either data-oriented (e.g. Jackson Structured Programming) or action-oriented (e.g. Data Flow Analysis), and therefore cannot capture the full complexity of agent-based systems that is characterised by the knowledge (data) and behaviour (actions) of individual agents *together with* their interaction.

Although structured methods do not provide fully adequate frameworks for specifying the complex dynamic interactions that take place in agent-based systems, they may be helpful in the early stages of designing heterogeneous systems that make use of complex deliberative agents. When working with this early high-level design phase, it may be that process and enterprise modelling

techniques could be seen as a possible resource for use in conjunction with structured methods to help with the design of large agent systems. However, in more complex agent-based systems, the need to represent and develop cooperative action is likely to mitigate against the use of traditional structured methods.

4 Using object-oriented methods?

In developing agent-based systems can we use variations on object-oriented methods?

As many of the systems termed "agent-based", particularly INTERNET agents, are closely related to object-based systems, popular analysis and design methods for object-based languages (e.g. Fusion) may be able to be used directly.

Of the methods which have been specifically applied to agent-based systems, Georgeff and Kinny's (1997) work is highly regarded. They set up a design methodology based on object-oriented analysis and design models (OOAD) and differentiate between an internal and an external view of an agent-based system. The internal view explains how an agent works internally and should be based on a well-defined operational model; the external view describes the agent–environment relationship and is split into two submodels, the agent model and the interaction model. Decoupling the two views allows us to develop agents internally based on a suitable architecture, while being able to analyse and design the system as a whole, independent from this architecture.

While using the OOAD paradigm as a basis for an agent-based methodology offers some important advantages, including that it is likely to be accepted as it is based on a well-known strategy, and that systems analysts and designers are likely to feel comfortable with the model as it naturally extends the OOAD model, it is clearly not sufficient. Such methods do not address issues of autonomy (e.g. decoupling the process of receiving a message from the action taken upon receiving it), reactivity and proactiveness. They also fail to address interaction on a higher level: what are the communication primitives that agents might use; what is their semantics; what are protocols and strategies employed in negotiation; how can cooperation be represented?

5 Using formal methods?

In developing agent-based systems, should we use variations on standard formal methods? Even if we do not utilise these explicitly, what can we learn from existing formal methods for concurrent systems?

Formal methods *may* be useful in developing agent-based systems, in particular when *critical* applications are being developed (this is obviously important in applications such as air traffic control, power station management, etc.), when prototyping agent-based systems at a high-level (since much of the detail can be hidden using logical statements and the key behaviour can be animated through executable specifications) and when developing complex cooperating systems (here the use of formal methods may be one way to establish that the system will exhibit the cooperative behaviour required).

Although traditional formal methods such as Z or VDM have been used in the specification of agent-based systems (Luck and d'Inverno, 1997), they are less well suited to specifying agent interactions, and there are a wide variety of formalisms that have been developed for concurrent and distributed systems (i.e. reactive systems). Since agent-based systems are essentially reactive systems, we should be able to transfer some of the techniques from concurrency theory (e.g. using CCS, CSP, temporal logics, etc) to agent-based systems, extending these basic approaches with the core features of agents such as autonomy and social ability.

While formal methods are not easily scalable in practice, due to the complexity of proof methods, they may be used to provide deeper understanding of certain critical parts of an agent-based system. Another important benefit of utilising standard formal approaches may be verification, which is well established in concurrency theory and provides a number of practical verification tools. Well-known problems such as the mutual-exclusion problem apply also to agent-based systems, whenever agents

have to access critical resources in a synchronized way. Concurrency theory may provide (useful ideas for) solutions of such problems.

Thus, formal methods are being applied to agent-based systems, often by extending formal approaches to concurrent systems, such as temporal logic, to incorporate the key elements of agents. However, it is again the case that the full range of agent capabilities have yet to be incorporated into a formally based development method. (See the report on the panel on *Formalisms for Multi-Agent Systems* for a more detailed discussion of formal methods for agent development.)

6 What do we require for development methods?

What is required of agent theories for them to be useful in the development of agent-based systems? In particular, what kinds of agent theories and programming models for agent-based systems lend themselves to principled development?

It is essential that the agent theory be an appropriate one for the modelling required. It is generally a mistake to worry too much about operational issues at a high level. Once the theoretical framework is in place, and it can be shown to model all the required behaviours, then consideration can be given to refinement and implementation. However, as Rodney Brooks has pointed out, many AI theories of knowledge representation have been mainly proposed to handle certain representation and reasoning problems, but they are never actually used (because they have not really been designed for practical use). These rather academic theories typically make conceptual and ontological stipulations which are not grounded in computational practice. Useful theories relate to the basic components and operations constituting their domain. They provide both a declarative and an operational semantics. If the latter is sufficiently elaborated, this already indicates that the theory will be a good basis for the development of working agent-based systems.

An agent-based representation, be it a theory or a language, should meet the following requirements: it must support both reactive and pro-active behaviour; it must have a formal semantics; the specification of agent behaviour should be both declarative and executable; it must be platform independent; it must be open; it must support heterogeneous agents; it must support the high-level notions of an agent's cooperative ability, its goal-directedness and its reflective capabilities; it must be modular. While agent theories must give a clear understanding of how an agent is defined in the respective theory (agent model) and how agents interact (interaction model), the programming language used must be able to represent these aspects in an appropriate way.

To evaluate agent theories and programming models, benchmark problems are obviously important. Once a benchmark is accepted, results are comparable and research becomes more competitive. It may be difficult, however, to identify good benchmarks and have them accepted by the agent-based systems community.

7 Bridging the gap between theory and practice

Although there has been important research in both agent theories and agent-based programming, the gulf between these is often large. Rather than producing development methods to bridge this gap, should we consider either providing theories at a lower level or programming languages at a higher level?

If possible, both of these should be provided; combining top down and bottom-up approaches is a good way to bridge gaps. Theories, starting from the kind of behaviour one would *like* to have, need to become more operational. Languages, usually developed by people that care very much about what one *can* do, need to take a step further in supporting higher-level agent-based concepts such as goals, plans, and capabilities. If a choice has to be made, then the provision of higher-level programming languages is essential. We should not compromise the high-level modelling of agent-based systems by forcing too many operational elements into agent theories.

Finally, a significant practical difficulty which occurs when one tries to work with agent-based systems is that the skills required to design and develop such systems run the whole gamut, from the

ability to work at very high-level designs and high-level programming through to the ability to work with detailed operating system and communication layers. As it is often very difficult to get those kind of skills together in small teams or individual efforts, the abstract modelling and development of systems without being concerned with the full detail of implementation would be particularly useful.

8 Tension between AI and computer science

Should we address artificial intelligence or software engineering concerns in agent-based systems? How is this different to mainstream computer science?

If we are considering building real systems, then it is difficult to see how we can (or would want to) avoid software engineering questions. If we intend to build software systems that serve any useful purpose we should utilise any useful technology. AI has much to offer as regards useful features and capabilities of the individual agent; distributed AI addresses models of cooperation while software engineering contains valuable guidelines for actually building systems.

Although, from a general point of view, we are doing much the same as other computer scientists (i.e. developing complex systems), we have specific ambitions. We want to build "programs with common-sense" (John McCarthy). Agent-based systems represent much of the grand vision of AI and they combine many important sub-disciplines. An agent has to reason, to plan, to act, etc. On the other hand we could be less ambitious, and just take the software engineering point of view considering agent-based systems as an extension of the object-oriented paradigm. Both approaches can learn from each other and may eventually converge: the former in a top-down, and the latter in a bottom-up fashion.

9 Agent testbeds

How important are testbeds for agent-based systems?

While testbeds, and testing regimes in general, are always useful in the production of complex, open, and reactive systems, they are particularly important in the development of agent-based systems. Not only are the systems developed potentially *very* complex, incorporating cooperation, competition and evolution, but the large numbers of agents used requires test suites exhibiting a large range of potential configurations.

There is a clear distinction between two uses of testbeds: a domain-related use and a technology-related use. The former simulates complex domains in order to gain empirical results allowing us to explain, predict, or verify their behaviour. Certainly, a common feature of agent-based systems is that they are often applied to model complex, open, and distributed domains. This is a valuable aspect of testbeds that is independent of what technology (agents, objects, etc.) is used to model the respective domain. The latter usage of testbeds involves an evaluation of the technology itself. In this respect, testbeds are suitable for the agent domain, since a good understanding of what agent-based systems are and how they behave is still lacking. Once we have this experience, agent testbeds of this sort lose their importance.

10 Current approaches

What are the current approaches, and what do they offer?

Though the area of development methods for agent-based systems is still very young, there are several approaches that are being used and investigated by the panel members. These are primarily concerned with the provision of languages and testbeds for the development of prototype systems.

10.1 VIVA knowledge-based agent programming

VIVA is a rule-based agent-oriented programming language (Wagner, 1996). It adopts many concepts from SQL and Prolog such as the distinction between the schema and the state of an agent, or the use of facts and rules with negation-as-failure, logical variables and unification. In addition to the well-known *deduction rules* of Prolog, VIVA also employs *action rules* for representing the pro-active behaviour repertoire, and *reaction rules* for representing the reactive behaviour of an agent (Schroeder and Wagner, 1997).

A first prototype interpreter (not yet suitable for distribution) has been implemented on the basis of PVM-Prolog. It is currently used to evaluate basic constructs of the language (Schroeder et al., 1997). A full-fledged VIVA interpreter is planned, using *Visual Prolog*, and which will be available for a variety of platforms.

10.2 SIM_AGENT

The Birmingham SIM_AGENT toolkit is intended to support exploration of designs in a variety of application domains, including, for example, adventure games involving a number of interacting characters, control mechanisms in which a number of different components concurrently monitor and send control signals to various parts of a complex machine or factory, teaching systems where intelligent agents interact with a trainee user, and for cognitive scientists wishing to explore designs for more or less human-like agents, including emotional agents. It is currently being used both at Birmingham University and at DRA Malvern. For an overview of the toolkit, see Sloman and Poli, (1996).

10.3 Concurrent METATEM

Concurrent METATEM (Fisher, 1993) is a language in which to represent agent-based systems (Fisher, 1995). It is dynamic (using temporal logic to represent individual agent behaviours), open (as broadcast message-passing is central and replication is achieved via cloning), flexible (through asynchronous concurrency, asynchronous message-passing and structuring of agent-space via groups), and simple (because that is all there is). The representations can either be executed directly (Fisher, 1996), verified with respect to a logical requirement, or transformed into a more refined representation.

Currently, the system consists of: an interpreter, which is mainly used to test extensions and applications, and thus is not suitable for wide distribution; several example systems, including simple reactive systems, concurrent theorem-provers, multi-agent planning and a (slightly simplified) version of the Procedural Reasoning Systems (PRS) (this can also be seen as a semantics for the PRS); semantics for Concurrent METATEM, together with a (prototype) formal development methodology for transforming Concurrent METATEM systems; and verification techniques for propositional systems. Future work will aim to develop a better implementation (to include a compiler), represent agents using knowledge and belief as well as temporal logics (Fisher and Wooldridge, 1997), and produce a full development framework from a single high-level agent to a cooperating multi-agent system.

11 Summary

Agent-based systems are different! While they may superficially appear to be standard concurrent object-based systems, they can potentially capture a much more complex range of behaviours than is generally seen using objects. In particular, these behaviours might relate to an agent's communication, its knowledge, beliefs or desires, its goals or intentions, and the range of cooperative acts that it might engage in. These high-level components ensure that any development method for agent-based systems is likely to be complex. Although such methods are still a long way from being

ready for widespread use, they are under active development and we are already beginning to see frameworks for the representation and development of simple agent-based systems in the research community.

References

Fisher, M, 1993, "Concurrent MetateM—A Language for modeling reactive systems" In: Bode, A, Reeve, M and Wolf, G, eds, *PARLE'93 Proceedings of Parallel Architectures, and Languages, Europe (Lecture Notes in Computer Science, 694)* 185–196, Springer-Verlag.

Fisher, M, 1995, "Representing and executing agent-based systems" In: Wooldridge, M and Jennings, N, eds, *Intelligent Agents — Proceedings of the 1994 Workshop on Agent Theories, Architectures, and Languages (Lecture Notes in Artificial Intelligence, 890)* 307–323, Springer-Verlag.

Fisher, M, 1996, "An introduction to executable temporal logics" *Knowledge Engineering Review* **11**(1) 43–56.

Fisher, M and Wooldridge, M, 1997, "On the formal specification and verification of multi-agent systems" *International Journal of Cooperating Information Systems* **6**(1).

Franklin, S and Graesser, A, 1997, "Is it an agent, or just a program?: A taxonomy for autonomous agents" In: Müller, J, Wooldridge, M and Jennings, N, eds, *Intelligent Agents III—Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (Lecture Notes in Artificial Intelligence, 1193)*, 21–35. Springer-Verlag.

Kinny, D and Georgeff, M, 1997, "Modelling and design of multi-agent systems" In: Müller, J, Wooldridge, M and Jennings, N, eds, *Intelligent Agents III—Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (Lecture Notes in Artificial Intelligence, 1193)*, 1–20. Springer-Verlag.

Luck, M, Griffiths, N and d'Inverno, M, 1997, "From agent theory to agent construction: A case study" In: Müller, J, Wooldridge, M and Jennings, N, eds, *Intelligent Agents III—Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (Lecture Notes in Artificial Intelligence, 1193)*, 49–63. Springer-Verlag.

Sloman, A and Poli, R, 1997, "SIM_AGENT: A toolkit for exploring agent designs" In: Wooldridge, M, Müller, J and Tambe, M, eds, *Intelligent Agents II—Proceedings of the Second International Workshop on Agent Theories, Architectures, and Languages (Lecture Notes in Artificial Intelligence, 1037)*, 392–407. Springer-Verlag.

Schroeder, M, Marques, R, Wagner, G and Cunha, J, 1997, "CAP—Concurrent Action and Planning: Using PVM-Prolog to Implement Vivid Agents" *Proceedings of the Fifth International Conference on The Practical Application of PROLOG*.

Schroeder, M and Wagner, G, 1997, "Distributed Diagnosis by Vivid Agents" *Proceedings of the First International Conference on Autonomous Agents* ACM Press.

Wagner, G, 1996, "VIVA Knowledge-Based Agent Programming" *Technical Report*, University of Leipzig, Germany.