The Construction of Multi-agent Systems as an Engineering Discipline

Jorge J. Gomez-Sanz

Universidad Complutense de Madrid,
Avda. Complutense s/n, 28040 Madrid, Spain
jjgomez@sip.ucm.es
http://grasia.fdi.ucm.es

Abstract. The construction of multi-agent systems is starting to become a main issue in agent research. Using the Computer Science point of view, the development of agent systems has been considered mainly a problem of elaborating theories, constructing programming languages implementing them, or formally defining agent architectures. This effort has allowed important advances, including a growing independence of Artificial Intelligence. The results have the potential to become a new paradigm, the agent paradigm. However, the acceptance of this paradigm requires its application in real industrial developments. This paper uses this need of addressing real developments to justify the use of software engineering as driving force in agent research. The paper argues that only by means of software engineering, a complex development can be completed successfully.

Keywords: agent oriented software engineering, multi-agent systems, development.

1 Introduction

Software agents have been considered as part of Artificial Intelligence (AI) for a long time. However, looking at the current literature, it seems agent researchers are focusing more on development aspects, such as debugging or modularity, paying AI issues less attention. This tendency is giving agent research an identity as a software system construction alternative, i.e., as a new paradigm. This paradigm will be named in this paper as the agent paradigm, whose existence was already pointed out by Jennings [1]. Informally, the agent paradigm is a development philosophy where main building blocks are agents.

In the elaboration of the agent paradigm, Computer Science has played a main role so far. With its aid, several agent oriented programming languages have been constructed and models of agency have been formalised. Nevertheless, it cannot be said the agent paradigm is mature, yet. Since agents are software, it is necessary proving the agent paradigm can be an alternative to known software development paradigms in a real, industrial, developments. This implies that the cost of developing a system using agents should be better than the cost of

G. O'Hare et al. (Eds.): ESAW 2006, LNAI 4457, pp. 25–37, 2007.

using other technologies, at least for some problems. Proving this goes beyond the scope of Computer Science and enters the domain of software engineering (it is assumed the readers accept Computer Science and Software Engineering are different disciplines).

This need of exploring the engineering side of software agents has been known in the agent community for some years. As part of this investigation, development environments have been created, debugging techniques have been elaborated, and agent oriented specification techniques have been proposed. These, together with the improvements in agent oriented programming languages, constitute important steps towards the integration with industrial practices [2]. Nevertheless, the growing incorporation of agents to industry demands a stronger effort.

The paper constributes with an argumentation in favour of applying engineering methods to achieve matureness in the agent paradigm. This argumentation starts with an introduction to the origins of software engineering in section 2. Then, section 3 introduces briefly how agents have started to focus on developments while being part of AI. These initial steps have led to results dealing with system construction that can be considered more independent of AI. These new results are introduced in section 4. Following, section 5 argues the need of engineering methods using the evolution of the agent research towards the construction of systems and drawing similitude with the origins of software engineering. This argumentation is followed, in section 6, by a suggestion of lines of work that could constribute to the acceptance of the agent paradigm. The paper finishes with some conclusions 7.

2 The Origins of Software Engineering

The claim of the paper requires understanding what is software engineering. The term *software engineering* was born in a NATO sponsored conference [3]. This conference invited reputed experts in software development to talk about the difficulties and challenges of building complex software systems. Many topics were discussed, but above all there was one called the *software crisis*. This was the name assigned to the increasing gap between what clients demanded and what was built due to problems during the development. Indeed, the smaller a development is, in the number of involved persons and in the complexity of the system to develop, the easier it is to complete successfully. Unfortunately, the software systems required in our societies are not small.

Dijkstra [4] had the opinion that a major source of problems in a development was the programming part. He remarked the importance of having proofs to ensure the correctness of the code, since much time is dedicated to debugging. Also, better code could be produced with better programming languages, like PL/I, and programming practices, like removing the goto statement from programming [5]. Dijkstra was right. Good programming and better programming languages reduce the risk of failure. Nevertheless, good code is not enough when programming large systems. For instance, De Remer [6] declares programming a large system with the languages at that time was an exercise in obscuration.

It was needed having a separated view of the system as a whole to guide the development, a view that was not provided by the structured programming approaches. In this line, Winograd [7] comments that systems are built by means of combining little programs. Hence, the problem is not building the small programs, but how the integration of packages and objects with which the system can be achieved.

Besides the programming, reasons for this gap can be found in the humans involved in a software project. Brooks [8] gives some examples of things that can go wrong in a development like being too optimistic about the effort a project requires, thinking that adding manpower makes the development going faster, or dealing with programmers wanting to exercise their creativity with the designs.

In summary, bridging the gap requires dealing with technical and human problems. Their solution implies defining a new combination of existing disciplines to build effectively software systems. This new discipline is the software engineering and their practitioners are called software engineers. The training software engineers receive is not the same of a a computer scientists [9]. This claim would be realised in the computing curricula recommendations by the ACM and IEEE for software engineering [10]. This proposal covers all aspects of a development, including programming and project management.

2.1 Lessons Learnt

There are three lessons to learn from the experience of the *software crisis* that will be applied to agent research in section 5.

The first lesson refers to the natural emergence of engineering proposals when the foundations are well established [9]. This has occurred many times with natural sciences and their application to human societies. As a result civil engineers, electrical engineers, and aerospace engineers, among others, exist.

The second is that some systems are inherently complex and require many people involved and an important amount of time. Building these systems is not a matter of sitting down in front of the computer and start working on it. It requires organising the work, monitoring the progress, and applying specific recipes at each step. Besides, there is a human component that cannot be fully controlled.

The third lesson is that the progress in programming languages is not going to eliminate all problems in a development. Three decades of progress in programming languages have made the construction of systems easier, but it has not eliminated the need of software engineering.

3 At the Beginning Everything Was Artificial Intelligence

The Artificial Intelligence discipline is a multi-disciplinary one dedicated to the creation of artificial entities capable of intelligent behaviour. The roots of agents are associated strongly to AI. Allen Newell made one of the first references to agents pointing out their relevance to AI. He distinguished agents as programs

existing at the *knowledge level* that used knowledge to achieve goals [11]. Two decades after, agents continue being relevant to AI. For instance, the Artificial Intelligence textbook from Russell and Norvig [12] presents agents as the embodiment of Artificial Intelligence.

Despite these statements, considering software agents only as AI would be a mistake. AI is important if certain features are to be incorporated in the agents, like planning or reasoning. However, the role of AI is not a dominant one and needs to be reviewed depending on the kind of system developed [2]. As a consequence, it is possible having an agent paradigm that can use AI while keeping its identity as paradigm. There have been similar cases in the past. Lisp [13] and Prolog [14] programming languages were born under the umbrella of AI. They are now independent and represent an important milestone in the establishment of the functional and logic programming paradigm, respectively. Following these precedents, one could draw the conclusion that focusing on particular approaches to the construction of AI capable systems has led to the definition of new development paradigms. Hence, the efforts in showing how a multi-agent system is built would eventually take to the definition of the agent paradigm.

Looking at the agent literature, published works dealing with multi-agent systems construction can be categorised into two groups: agent oriented programming languages and agent architectures. In both, Computer Science has played an important role, specially in agent oriented programming languages. Hence, most of these steps are founded on formal descriptions and theories.

The invention of the agent oriented programming language is usually dated in 1993 with the creation of Agent0 [15]. This language proposed using agents as building blocks of a system, providing a vocabulary to describe what existed inside of the agents and some semantics of how these agents operated. This trend has been explored from a computer science perspective, mainly, trying to establish theoretical principles of agent oriented programming. This has been the case of Jason [16] an extended interpreter for AgentSpeak (L) language [17], 3APL [18] or Agent Factory - APL [19] a combination of logic and imperative programming, Claim [20] which is based on ambient calculus, ConGOLOG [21] which bases on situation calculus, or Flux [22] which is based on fluent calculus, to cite some. For a wider review about agent oriented languages, readers can consult the short survey [23], the survey of logic based languages [24], or the more detailed presentation of programming languages from [25]. The diversity of concepts and approaches evidences there is still struggle to find the best way of dealing with agent concepts.

The definition of agent architectures is the other way identified to construct agents. An agent architecture is a software architecture that explains how an agent is built. A software architecture, according to IEEE, is *the fundamental organisation of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution* [26]. This way of defining systems was applied early on AI to explain how systems were capable of intelligent behaviour. In most cases, the agent architecture has been a set of boxes with arrows representing the data flow. The computations

performed inside the boxes were explained in natural language. Examples of these initial architectures are the reference architecture for adaptative intelligent systems [27], the Resource-bounded practical reasoning architecture [28], or Archon [29]. This kind of architectures were not fully detailed, but they provided important hints to a developer wanting to reuse them. These architectures provided a set of elements to look at and a way of organising them. A higher level of detail would be given by Interrap [30] and Touring Machines [31]. Interrap provided formal definitions of the inputs and outputs of the different elements of the architecture. Touring machines provided a reference architecture and detailed instructions to fill in each component of the architecture with Sicstus Prolog. Its capabilities were demonstrated in a real time traffic management scenario with dynamic changes. Perhaps, the most complete architecture described so far is DESIRE [32]. The semantics of DESIRE are formally specified using temporal logic [33] and allowed compositional verification [34]. Besides formal aspects, DESIRE differs from previous ones in how it was used. Instead of providing guidelines, it used a formal language to describe how the architecture was instantiated to produce a system. This language was processed by ad-hoc tools to produce the an instance of DESIRE for the current problem domain. For a wider review of architectures, readers can consult the surveys [35] and [36].

4 Focusing on Multi-agent Systems Construction

The examples presented in previous sections demonstrate an increasing concern about the construction of systems by means of agent architectures and agent oriented programming languages. On one hand, agent oriented programming languages implement theories of agency devising new ways of processing information. On the other hand, agent architectures explore the internal components of an agent, often letting the developer choosing how these components are implemented. These precedents have given the opportunity of focusing more on development problems and less on AI, in concrete, the experience gained allowed to propose agent standards, agent development environments, and methodologies.

The creation of a standard, FIPA, and the availability of a platform implementing it, the JADE framework [37], turned out to be an important factor that increased the number of developments using agents. Agents no more had to be created from scratch. By extending some classes from the framework, one could have a simple multi-agent system. However, JADE provided very basic support for defining the behaviour of the agent. Most of the times, developers had to build over a JADE layer other layers dedicated to implement the control of the agent. FIPA work continues under the scope of IEEE, so there is still much to say.

With respect development environments to produce agent software, the first was the Zeus tool [38]. It offered a graphic front end that allowed to define the internals of the agents (ontology, action rules, goals), communication, and deployment of the system. The information introduced served to instantiate a multi-agent system framework, which could be customised by the developer. The main contribution of Zeus was the reduction of the development effort.

Little programming was required unless the generated system shown a bug in the automatically generated part. This development environment was followed by others like JADEX, Agent Factory or AgentTool [23]. These frameworks can be considered the inheritors of the first efforts in the definition of agent architectures. They still define the system in terms of components and relationships among them, but require less information from the domain problem to produce a tentative system.

Agent oriented methodologies contributed with support tools, proposals for the specifications of agent oriented systems, steps to follow to produce these specifications as well as to implement them. One of the first was Vowel Engineering [39]. It conceived the system as the combination of five interdependent aspects: Agent, Environment, Interaction, and Organisation. Another important contribution is the adaptation of BDI theory into an iterative development process [40]. Both proposals introduced a simple idea: constructing a multi-agent system could be less an art and more a discipline by itself. This was introduced later on by [1] describing several distinguishing features of agent computing that could constitute a new approach to software engineering. The next step in methodologies was MESSAGE [41] [42]. MESSAGE provided a comprehensive collection of agent concepts, supported by tools, and covering the complete development process. It was the first in applying meta-modelling techniques in an agent oriented methodology. Later on this would be incorporated to all methodologies. For a review of agent oriented methodologies, readers can read [43]. This book has a chapter dedicated each one of ten agent oriented methodologies. It include a comparison among them using an evaluation framework. This framework evaluates the existence and quality of a set of features that are desirable in a methodology.

5 The Need of Engineering Methods

The results presented in previous sections constribute to the understanding of the agent paradigm in different ways. In general, they make a development simpler. Nevertheless, the impact of those results has not been the one expected in industry. Agentlink conducted a study of the impact and evolution of agent technology in Europe during several years. The result is the AgentLink roadmap [44]. This study shows relatively few histories of success of agent technology in real developments. There may be reasons for this like slow penetration of new technologies in the market or companies being reluctant to tell what kind of software they use. However, it seems more reasonable to find answers in the way agent research has been directed. Zambonelli and Omicini [45] tell there is little effort in generating evidences of the savings in money and resources agents can bring. Luck, McBurney, and Priest [46] provide other reasons: *most platforms are still too immature for operational environments*, *lack of awareness of the potential applications of agent systems*, *the cost of system development and implementation both in direct financial terms and in terms of required skills and timescales*, or *the absence of a migration path* from agent research to industrial applications.

Drawing a parallelism with the situation presented in section 2, there is a particular *software crisis* affecting the agent community. This crisis relates to the capability of agent research to build applications in an effective way. The crisis can be summarised as *a gap between the application a client demands and what can be actually built with existing agent results according to the constraints of a development project.* Hence, for a greater acceptance of the agent paradigm, this paper proposes the agent community a new goal consisting in the reduction of this gap. The answer to the *software crisis* was software engineering, i.e., using engineering approaches to the construction of software. Therefore, the agent community ought to focus more on engineering issues as well. The call for more engineering is not new. Concretely, readers can find a precedent in [45].

Assuming the need of software engineering, it is relevant to review the lessons from section 2.1 in the context of agent research.

The first lesson talked about the natural emergence of engineering methods when sufficient foundations exist. There exist a need of developing agent oriented systems, and the results (languages, architectures, tools, theories) to do so exist. Therefore, an engineering discipline specialised in the use of these elements to produce agent oriented systems makes sense. This discipline would propose combinations of theoretical and practical agent reseach results so that developers can solve problems effectively.

The second lesson commented on the resources needed in a development of a complex system. Developing a complex system requires investing time and people, even if software agents are used. Therefore, developers should know the particularities of using agents when there is a deadline to finish the product, a maximum available budget, and several involved workers. This covers the requirements gathering, analysis, design, implementation, testing, and maintenance stages of a development. Known development roles will have to be revisited to look for changes due to the use of the agent paradigm. For instance, it is required to assist project managers with guidelines, cost estimation methods, risk management, and other common project management tools customised with the experience of successfull application of agent results.

The third lesson was about the need of complementing programming languages with other techniques. Agent research is not going to produce any magic bullet for complex system development. The agent paradigm will provide means comparable to other existing ones in the capability of building a system. The difference will be in the balance between results obtained and the effort invested. Should this balance be positive and greater than applying other paradigms, the agent paradigm will become a valid alternative. There are many factors to consider in the equation, but it will depend ultimately on the developers. So, in order to ensure a successfull development, it is necessary to provide all kind of support to these developers. This includes development tools, training, and documentation, to cite some.

These lessons tell the agent paradigm has to produce many basic engineering results, yet. Applying engineering is not only producing a methodology. It is about, for instance, discovering patterns of recurrent configurations when using

agent oriented programming languages, designing agent architectures attending to the non-functional requirements, evaluating the cost of a agent oriented development, producing testing cases for agent applications, documenting properly a development, and so on. Agent oriented methodologies, which are not so recent, do not consider most of these elements.

6 Proposed Lines of Research

This section collects several lines of work in line with the lessons from section 5. Some lines have been started already, but require further investigation. Others have not been initiated, yet.

The main line consists in performing large developments. In order to bridge the gap pointed out in section 5, agent research must experiment with large developments involving several developers and dealing with complex problems. The benefit for agent research would be twofold. Besides gaining experience and validating development techniques, these developments would produce useful prototypes demonstrating the benefits of the agent paradigm. There are some specific areas where the development of multi-agent system prototypes could attract additional attention from industry. Luck, McBurney, and Priest [46] identify the following: ambient intelligence, bioinformatics and computational biology, grid computing, electronic business, and simulation. Ideally, the prototypes should be built by developers not involved directly in the research of the techniques applied. Like in software tests [47], if the tester is the creator of the technique, the evaluation could contain some bias.

The results of these large developments would influence in the existing agent frameworks, agent development environments, and agent oriented programming languages. These should be revisited in the light of the experience from large developments. Expected improvements derived from this experience will improve the debugging techniques (it is likely to invest an important time detecting where the failure is), the quality of code (good programming practices for agent oriented programming languages), and the code reuse (large developments will force developers to find ways to save coding effort).

Also, the development of these prototypes could give opportunities to experience other aspects not documented yet in the agent literature. Project management and maintenance are two of them. Inside project management, project planning and risk management, two key activities in project management, have not been experienced when agents are applied. Cost estimation, which is relevant for planning, is almost missing in agent research. Gomez-Sanz, Pavon, and Garijo [48] present some seminal work in this direction using information extracted from European projects. With respect maintenance, the maintenability of code written with an agent oriented programaming language, agent frameworks, or an agent architecture is a big unknown at this moment. The cost of correcting defects or adding new functionality once the system is deployed is unknown as well.

With respect to project deliverables, it is a matter of discussion to determine which deliverables are needed in an agent oriented project beyond those known of software engineering. Deliverables determines the progress the project should have. Hence, defining a proper content of these deliverables is a priority for a project manager. From these deliverables, the only one being studied currently is the system specification. It is being defined with meta-models, a technique imported from software engineering. The technique has found quickly a place in agent research. In fact, most existing methodologies have a meta-model description for system specification. Agent researchers intend to unify the different existing meta-models [49], though this is a work that will take time.

The particularities of agent oriented development process are starting to be investigated. Brian Henderson-Sellers [50] [51] contributes with a metamodel to capture development methods and represent existing agent oriented development approaches. Cernuzzi, Cossentino, and Zambonelli [52] studies the different development process to identify open issues.

There is concern about the importance of debugging. Development environments like JADEX and Agent Factory provide basic introspection capabilities. Also, Botia, Hernansaez, and Skarmeta[53] study how to debug multi-agent systems using ACL messages as information source. Lam and Barber try to explain why the agent behaved in a concrete way using information extracted from traces of the system [54]. Testing is an activity less studied.

Code generation may turn out to be a key to the success of agents. Most of the works are agent frameworks which has to be instantiated sometimes with a formal language, others with a visual environment. Generating automatically the instantiation parameters should not be a hard task. There are precedents of this kind of tools in Zeus [38], AgentTool [55], and the INGENIAS Development Kit [56]. The benefits are clear, since the developer invests little effort and obtains a working system in exchange. However, there are drawbacks in the approach, like conserving the changes made in the generated code.

To conclude, there is a line of research related with software engineering that has not been considered yet. Besides structuring an agent oriented development, the effort invested in the creation of methodologies can be used to start the creation of bodies of knowledge for agent oriented software engineering. In software engineering, there exist a body of knowledge [57] that provide an agreed terminology for practitioners. This initiative was started in a joint effort by the IEEE and ACM in order to develop a profession for software engineers. In this line, readers can consult an extended review of agent research results that can be useful in each concrete stage of a development [58].

7 Conclusions

Agent research will eventually gain more independency from Artificial Intelligence. This independency will be completely realised with the establishment of the agent paradigm, the development philosophy that uses agents as main building blocks. According to recent studies, the adoption of agent research re-

sults by industry, which would be an evidence of the matureness of the agent paradigm, is not progressing as it should. A reason for this delay may be the lack of large developments that prove the benefits of choosing the agent paradigm. Agent research has been driven mainly by Computer Science focusing in concrete features of the agents and the experimentation has been limited to small developments, mainly. As a result, there is no enough experience in large developments of software systems based on agents, so the question of the capability of agent technology to deal with a real development remains. This problem has been presented as a kind of *software crisis* of the agent community.

To deal with this crisis, this paper has proposed the extensive use of software engineering principles. This requires prioritising the research on aspects that have not been considered before in an agent community, like project management issues, good practices of agent oriented programming, or maintance costs associated to a multi-agent system. To illustrate the kind of areas to address, the paper has pointed at several lines of work, referring to some preliminary work already done in those directions.

Acknowledgements. This work has been supported by the project Methods and tools for agent-based modelling supported by Spanish Council for Science and Technology with grant TIN2005-08501-C03-01, and by the grant for Research Group 910494 by the Region of Madrid (Comunidad de Madrid) and the Universidad Complutense Madrid.

References

1. Jennings, N.: On agent-based software engineering. Artificial Intelligence 117(2), 277–296 (2000)
2. Dastani, M., Gomez-Sanz, J.J.: Programming multi-agent systems. The Knowledge Engineering Review 20(02), 151–164 (2006)
3. Naur, P., Randell, B. (eds.): Software Engineering: report on a conference sponsored by the nato science committee, Garmisch, Germany, NATO Science Committee (1968)
4. Dijkstra, E.: The humble programmer. Communications of the ACM 15(10), 859–866 (1972)
5. Dijkstra, E.: Goto statement considered harmful. Communications of the ACM 11(3), 147–148 (1968)
6. DeRemer, F., Kron, H.: Programming-in-the large versus programming-in-the-small. In: Proceedings of the international conference on Reliable software, pp. 114–121. ACM Press, New York (1975)
7. Winograd, T.: Beyond programming languages. Communications of the ACM 22(7), 391–401 (1979)
8. Brooks, F.: The mythical man-month: essays on software engineering. Addison-Wesley, London, UK (1982)
9. Parnas, D.: Software engineering programmes are not computer science programmes. Annals of Software Engineering 6(1), 19–37 (1998)

10. Diaz-Herrera, J.L., Hilburn, T.B.: SE 2004 - Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. In: The Joint Task Force on Computing Curricula IEEE Computer Society Association for Computing Machinery. IEEE Computer Society Press, Los Alamitos (2004)
11. Newell, A.: The knowledge level. Artificial Intelligence 18, 87–127 (1982)
12. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice-Hall, Englewood Cliffs (2003)
13. MacCarthy, J.: Lisp 1.5 Programmer's Manual. MIT Press, Cambridge (1965)
14. Roussel, P.: PROLOG: Manuel de reference et d'utilisation. Université d'Aix-Marseille II (1975)
15. Shoham, Y.: Agent-Oriented Programming. Artificial Intelligence 60(1), 51–92 (1993)
16. Bordini, R., Hübner, J., Vieira, R.: Jason and the Golden Fleece of Agent-Oriented Programming. In: Multi-Agent Programming. Multiagent Systems, Artificial Societies, and Simulated Organizations, vol. 15, pp. 3–37. Springer, Heidelberg (2005)
17. Rao, A.S.: AgentSpeak (L): BDI agents speak out in a logical computational language. In: Perram, J., Van de Velde, W. (eds.) MAAMAW 1996. LNCS, vol. 1038, pp. 42–55. Springer, Heidelberg (1996)
18. Hindriks, K., Boer, F.D., der Hoek, W.V., Meyer, J.: Agent Programming in 3APL. Autonomous Agents and Multi-Agent Systems 2(4), 357–401 (1999)
19. Ross, R.J., Collier, R.W., O'Hare, G.M.P.: Af-apl - bridging principles and practice in agent oriented languages. In: Bordini, R.H., Dastani, M., Dix, J., Seghrouchni, A.E.F. (eds.) Programming Multi-Agent Systems. LNCS (LNAI), vol. 3346, pp. 66–88. Springer, Heidelberg (2005)
20. Fallah-Seghrouchni, A.E., Suna, A.: CLAIM: A Computational Language for Autonomous, Intelligent and Mobile Agents. In: Dastani, M., Dix, J., El Fallah-Seghrouchni, A. (eds.) PROMAS 2003. LNCS (LNAI), vol. 3067, pp. 90–110. Springer, Heidelberg (2004)
21. Shapiro, S., Lesperance, Y., Levesque, H.: Specifying communicative multi-agent systems with ConGolog. In: Working Notes of the AAAI Fall 1997 Symposium on Communicative Action in Humans and Machines, vol. 1037, pp. 72–82 (1997)
22. Thielscher, M.: FLUX: A logic programming method for reasoning agents. Theory and Practice of Logic Programming 5(4-5), 533–565 (2005)
23. Bordini, R., Braubach, L., Dastani, M., Seghrouchni, A.E.F., Gomez-Sanz, J., Leite, J., O'Hare, G., Pokahr, A., Ricci, A.: A Survey of Programming Languages and Platforms for Multi-Agent Systems. Informatica 30(1), 33–44 (2006)
24. Mascardi, V., Martelli, M., Sterling, L.: Logic-based specification languages for intelligent software agents. Theory and Practice of Logic Programming Journal (2004)
25. Bordini, R.H., Dastani, M., Dix, J., Seghrouchni, A.E.F. (eds.): Multi-Agent Programming. Multiagent Systems, Artificial Societies, and Simulated Organizations, vol. 15. Springer, Heidelberg (2005)
26. Hillard, R.: Recommended practice for architectural description of software-intensive systems. Technical report, IEEE (2000)
27. Hayes-Roth, B., Pfleger, K., Lalanda, P., Morignot, P., Balabanovic, M.: A domain-specific software architecture for adaptive intelligent systems. IEEE Transactions on Software Engineering 21(4), 288–301 (1995)
28. Bratman, M.E., Israel, D.J., Pollack, M.: Plans and resource-bounded practical reasoning. Computational Intelligence Journal 4(4), 349–355 (1988)

29. Jennings, N.R., Wittig, T.: ARCHON: Theory and Practice. In: Distributed Artificial Intelligence: Theory and Praxis. Eurocourses: Computer and Information Science, vol. 5. Springer, Heidelberg (1992)
30. Müller, J., Pischel, M.: The Agent Architecture InteRRaP: Concept and Application. PhD thesis, Deutsches Forschungszentrum für Künstliche Intelligenz (1993)
31. Ferguson, I.: Touring Machines: An architecture for Dynamic, Rational Agents. PhD thesis, Ph. D. Dissertation, University of Cambridge, UK (1992)
32. Brazier, F., Dunin-Keplicz, B., Jennings, N., Treur, J.: DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework. International Journal of Cooperative Information Systems 6(1), 67–94 (1997)
33. Brazier, F., Treur, J., Wijngaards, N., Willems, M.: Temporal semantics of complex reasoning tasks. In: Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-based Systems workshop, vol. 96, pp. 1–15 (1996)
34. Cornelissen, F., Jonker, C., Treur, J.: Compositional Verification of Knowledge-based Systems: a Case Study for Diagnostic Reasoning. In: Plaza, E. (ed.) EKAW 1997. LNCS, vol. 1319, pp. 65–80. Springer, Heidelberg (1997)
35. Wooldridge, M., Jennings, N.: Agent Theories, Architectures, and Languages: A Survey. Intelligent Agents 22 (1995)
36. Wooldridge, M., Jennings, N.R.: Intelligent agents: Theory and practice. Knowledge Engineering Review 10(2), 115–152 (1995)
37. Bellifemine, F., Poggi, A., Rimassa, G.: Jade: a fipa2000 compliant agent development environment. In: AGENTS '01: Proceedings of the fifth international conference on Autonomous agents, pp. 216–217. ACM Press, New York (2001)
38. Nwana, H.: Zeus: A Toolkit for Building Distributed Multi-agent Systems. Applied Artificial Intelligence 13(1), 129–185 (1999)
39. Demazeau, Y.: From interactions to collective behaviour in agent-based systems. In: Proceedings of the 1st. European Conference on Cognitive Science.
40. Kinny, D., Georgeff, M., Rao, A.: A methodology and modelling technique for systems of BDI agents. In: Perram, J., Van de Velde, W. (eds.) MAAMAW 1996. LNCS, vol. 1038, pp. 56–71. Springer, Heidelberg (1996)
41. Caire, G., Coulier, W., Garijo, F.J., Gomez, J., Pavón, J., Leal, F., Chainho, P., Kearney, P.E., Stark, J., Evans, R., Massonet, P.: Agent oriented analysis using message/uml. In: Wooldridge, M.J., Weiß, G., Ciancarini, P. (eds.) AOSE 2001. LNCS, vol. 2222, pp. 119–135. Springer, Heidelberg (2002)
42. Evans, R., Kearney, P., Caire, G., Garijo, F., Gomez-Sanz, J.J., Pavon, J., Leal, F., Chainho, P., Massonet, P.: Message: Methodology for engineering systems of software agents (September 2001), `http://www.eurescom.de/~pub-deliverables/p900-series/P907/TI1/p907ti1.pdf`
43. Henderson-Sellers, B., Giorgini, P.: Agent-oriented methodologies. Idea Group Pub. USA (2005)
44. Luck, M., McBurney, P., Shehory, O., Willmott, S.: Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing). AgentLink (2005)
45. Zambonelli, F., Omicini, A.: Challenges and Research Directions in Agent-Oriented Software Engineering. Autonomous Agents and Multi-Agent Systems 9(3), 253–283 (2004)
46. Luck, M., McBurney, P., Preist, C.: A Manifesto for Agent Technology: Towards Next Generation Computing. Autonomous Agents and Multi-Agent Systems 9(3), 203–252 (2004)
47. Myers, G., Sandler, C., Thomas, T.M., Badgett, T.: The Art of Software Testing. John Wiley and Sons, West Sussex, England (2004)

48. Gomez-Sanz, J., Pavon, J., Garijo, F.: Estimating Costs for Agent Oriented Software. In: Müller, J.P., Zambonelli, F. (eds.) AOSE 2005. LNCS, vol. 3950, pp. 218–230. Springer, Heidelberg (2006)
49. Bernon, C., Cossentino, M., Pavon, J.: Agent-oriented software engineering. The Knowledge Engineering Review 20(02), 99–116 (2006)
50. Gonzalez-Perez, C., McBride, T., Henderson-Sellers, B.: A Metamodel for Assessable Software Development Methodologies. Software Quality Journal 13(2), 195–214 (2005)
51. Henderson-Sellers, B.: Creating a Comprehensive Agent-Oriented Methodology: Using Method Engineering and the OPEN Metamodel. In: Agent-Oriented Methodologies, pp. 368–397. Idea Group, USA (2005)
52. Cernuzzi, L., Cossentino, M., Zambonelli, F.: Process models for agent-based development. Engineering Applications of Artificial Intelligence 18(2), 205–222 (2005)
53. Botía, J.A., Hernansaez, J.M., Skarmeta, F.G.: Towards an approach for debugging mas through the analysis of acl messages. In: Lindemann, G., Denzinger, J., Timm, I.J., Unland, R. (eds.) MATES 2004. LNCS (LNAI), vol. 3187, pp. 301–312. Springer, Heidelberg (2004)
54. Lam, D.N., Barber, K.S.: Comprehending agent software. In: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, pp. 586–593. ACM Press, New York (2005)
55. DeLoach, S., Wood, M.F.: Developing multiagent systems with agenttool. In: Castelfranchi, C., Lespérance, Y. (eds.) ATAL 2000. LNCS (LNAI), vol. 1986, pp. 46–60. Springer, Heidelberg (2001)
56. Pavon, J., Gomez-Sanz, J.J., Fuentes, R.: The INGENIAS Methodology and Tools. In: Agent-Oriented Methodologies, pp. 236–276. Idea Group Publishing, USA (2005)
57. Bourque, P., Dupuis, R., Abran, A., Moore, J., Tripp, L.: The guide to the Software Engineering Body of Knowledge. Software 16(6), 35–44 (1999)
58. Gomez-Sanz, J.J., Gervais, M.P., Weiss, G.: A Survey on Agent-Oriented Software Engineering Research. In: Methodologies and Software Engineering for Agent Systems, pp. 33–62. Kluwer Academic Publishers, Dordrecht (2004)