A Survey of Agent-Oriented Methodologies*

Carlos A. Iglesias[1]**, Mercedes Garijo[2] and
José C. González[2]

[1] Dep. TSC e Ing. Telemática, E.T.S.I. Telecomunicación
Universidad de Valladolid, E–47011 Valladolid, Spain
cif@tel.uva.es
[2] Dep. de Ingeniería de Sistemas Telemáticos, E.T.S.I. Telecomunicación
Universidad Politécnica de Madrid, E–28040 Madrid, Spain
{mga,jcg}@gsi.dit.upm.es

Abstract. This article introduces the current agent-oriented methodologies. It discusses what approaches have been followed (mainly extending existing object-oriented and knowledge engineering methodologies), the suitability of these approaches for agent modelling, and some conclusions drawn from the survey.

1 Introduction

Agent technology has received a great deal of attention in the last few years and, as a result, the industry is beginning to get interested in using this technology to develop its own products. In spite of the different developed agent theories, languages, architectures and the successful agent-based applications, very little work for specifying (and applying) techniques to develop applications using agent technology has been done. The role of agent-oriented methodologies is to assist in all the phases of the life cycle of an agent-based application, including its management.

This article reviews the current approaches to the development of an agent-oriented (AO) methodology. To avoid building a methodology from scratch, the researchers on agent-oriented methodologies have followed the approach of extending existing methodologies to include the relevant aspects of the agents. These extensions have been carried out mainly in two areas: object oriented (OO) methodologies (Section 2) and knowledge engineering (KE) methodologies (Section 3). We will review (1) why each area can be relevant for developing an AO methodology, (2) what problems were found in applying directly the existing methodologies without extending them and (3) what solutions have been proposed. We will also review some particular approaches (Section 4), formal approaches (Section 5) and software-engineering techniques proposed by agent researchers (Section 6). Finally, some conclusions are drawn (Section 7).

* This research is funded in part by the Commission of the European Community under the ESPRIT Basic Research Project *MIX: Modular Integration of Connectionist and Symbolic Processing in Knowledge Based Systems*, ESPRIT-9119, and by the Spanish Government under the CICYT projects TIC91-0107 and TIC94-0139

** This research was partly carried out while the first author was visiting the Dep. Ingeniería de Sistemas Telemáticos (Universidad Politécnica de Madrid).

2 Extensions of Object-Oriented Methodologies

2.1 Advantages of the approach

Several reasons can be cited that justify the approach of extending object-oriented methodologies.

Firstly, there are similarities between the object-oriented paradigm and the agent-oriented paradigm [5, 27]. Since the early times of distributed artificial intelligence (DAI), the close relationship between DAI and Object-Based Concurrent Programming (OBCP) [2, 15] was established. As stated by Shoham [43], the agents can be considered as *active objects*, objects with a mental state. Both paradigms use message passing for communicating and can use inheritance and aggregation for defining its architecture. The main difference [43] is the constrained type of messages in the AO paradigm and the definition of a state in the agent based on its beliefs, desires, intentions, commitments, etc.

Another possible advantage comes from the *commonly usage* of object-oriented languages to implement agent-based systems because they have been considered a natural framework [2, p. 34].

The *popularity* of object-oriented methodologies is another potential advantage. Many object-oriented methodologies are being used in the industry with success such as Object Modelling Technique (OMT) [41], Object Oriented Software Engineering (OOSE) [21], Object-Oriented Design [3], RDD (Responsibility Driving Design) [48] and Unified Modelling Language (UML) [8]. This experience can be a key to facilitate the integration of agent technology because on the one hand, the software engineers can be reluctant to use and learn a complete new methodology, and on the other hand, the managers would prefer to follow methodologies which have been successfully tested. So we could take advantage of their experience for learning quicker.

The three common views of the system in object-oriented methodologies are also interesting for describing agents: *static* for the object structure objects and their structural relationships; *dynamic* for describing the object interactions; and *functional* for describing the data flow of the methods of the objects.

Finally, some of the techniques for object identification can also be used for identifying agents: use cases [21] and classes responsibilities collaborations (CRC) cards [48].

2.2 Aspects not addressed

In spite of the similarities between objects and agents, obviously, agents are not simply objects. Thus, object-oriented methodologies do not address these different aspects [43, 5, 24].

Firstly, though both objects and agents use message-passing to communicate with each other, while message-passing for objects is just method invocation, agents distinguish different types of messages and model these messages frequently as speech-acts and use complex protocols to negotiate. In addition, agents analyse these messages and can decide whether to *execute* the requested action.

Another difference consists in that agents can be characterised by their mental state, and object-oriented methodologies do not define techniques for modelling how the agents carry out their inferences, their planning process, etc.

Finally, agents are characterised by their social dimension. Procedures for modelling these social relationships between agents have to be defined.

2.3 Existing solutions

In this section the following agent-oriented methodologies are reviewed: *Agent-Oriented Analysis and Design* [5], *Agent Modelling Technique for Systems of BDI agents* [27], *MASB* [31, 32] and *Agent Oriented Methodology for Enterprise Modelling* [24].

Agent-Oriented Analysis and Design by Burmeister

Burmeister [5] defines three models for analysing an agent system: the *agent model*. that contains the agents and their internal structure (beliefs, plans, goals,...); the *organisational model*, that describes the relationships between agents (inheritance and roles in the organisation); and the *cooperation model*, that describes the interactions between agents.

The process steps for the development of each model are:

- *Agent Model:* agents and their environment are identified using an extension of CRC cards for including beliefs, motivations, plans and cooperation attributes.
- *Organisational Model:* proposes the identification of the roles of each agent and the elaboration of diagrams using OMT notation for the inheritance hierarchy and the relationships between the agents.
- *Cooperation Model:* cooperations and cooperation partners are identified, and the types of interchanged messages and used protocols are analysed.

Agent Modelling Technique for Systems of BDI agents

This method [27] defines two main levels (external and internal) for modelling BDI (Belief, Desire and Intention) agents.

The *external viewpoint* consists of the decomposition of the system into agents and the definition of their interactions. This is carried out through two models: the *agent model*, for describing the hierarchical relationship between agents and relationships between concrete agents; and the *interaction Model* [26], for describing the responsibilities, services and interactions between agents and external systems.

The *internal viewpoint* carries out the modelling of each BDI agent class through three models: the *belief model*, which describes the beliefs about the environment; the *goal model*, which describes the goals and events an agent can adopt or respond to; and the *plan model*, which describes the plans an agent can use to achieve its goals.

The development process of the *external viewpoint* starts with the identification of the roles (functional, organisational, etc.) of the application domain in order to identify the agents and arrange them in an agent class hierarchy described using OMT like notation. Then the responsibilities associated to each role are identified, together with the services provided and used to fulfill the responsibilities. The next step is the identification of the necessary interactions for each service and both the speech-act and information content of every interaction. Finally, the information is collected in an *agent instance model*.

The development of the *internal viewpoint* starts with the analysis of the different means (plans) of achieving a goal. The plans for responding to an event or achieving a goal are represented using a graphical notation similar to Harel statecharts [18], but adding the notion of failure of the plan. Finally, the beliefs of the agent about the objects of the environment are modelled and represented using OMT notation.

Multi-Agent Scenario-Based Method (*MASB* method)

This method [31, 32] is intended to be applied for MAS in the field of cooperative work (CSCW). The analysis phase consists of the following activities:

- *Scenario description:* identification using natural language of the main roles played by both the human and the software agents, objects of the environment and the typical scenarios.
- *Role functional description:* description of the agent roles using *behaviour diagrams*, which describe the processes, the relevant information and the interactions between the agents.
- *Data and world conceptual modelling:* modelling of the data and knowledge used by the agent using entity-relationship diagrams (or object-oriented diagrams) and entity life-cycle diagrams.
- *System-user interaction modelling:* simulation and definition of different suitable interfaces for human-machine interaction in every scenario.

The design phase consists of the following activities:

- *MAS architecture and scenario description:* selection of the scenarios to be implemented and the roles played by the agents in these scenarios.
- *Object modelling:* refines the world modelling of the analysis, defining hierarchies, attributes and procedures.
- *Agent modelling:* specification of the elements defined in the data conceptual modelling step of the analysis as belief structures. A graphical notation is proposed for describing the decision process of a agent, taking into account beliefs, plans, goals and interactions.
- Finally, two steps are stated though not developed: *conversation modelling* and *system design overall validation*.

Agent oriented methodology for Enterprise modelling

This methodology [24] proposes the combination of object-oriented methodologies (OOSE) and enterprise modelling methodologies IDEF (Integration DEfinition for Function modelling) [12] and CIMOSA (Computer Integrated Manufacturing Open System Architecture) [28]). The identified models are:

- *Function Model:* describes the functions (inputs, outputs, mechanisms and control) using $IDEF_0$ diagrams that include the selection of the possible methods depending on the input and the control.

– *Use Case Model:* describes the actors involved in each function, using OOSE use case notation.
– *Dynamic Model:* this model is intended for analysing object interactions. The use cases are represented in event trace diagrams.
– *The Agent Oriented System:* is a compound of:
  - *Agent Identification:* the actors of the use cases are identified as agents. The main functions of an agent are its goals and the possibilities described in the $IDEF_0$ diagrams.
  - *Coordination protocols or scripts:* they are described in state diagrams.
  - *Plan invocation: sequence diagrams* extend event trace diagrams to include conditions for indicating when a plan is invoked.
  - *Beliefs, Sensors and Effectors:* inputs of the functions should be modelled as beliefs or obtained from objects via sensors, and achieved goals should be modelled as changes to beliefs or modifications via effectors.

3 Extensions of Knowledge Engineering Methodologies

3.1 Advantages of the approach

Knowledge engineering methodologies can provide a good basis for MAS modelling since they deal with the development of knowledge based systems. Since the agents have cognitive characteristics, these methodologies can provide the techniques for modelling this agent knowledge.

The definition of the knowledge of an agent can be considered as a *knowledge acquisition process*, and only this process is addressed in these methodologies.

The extension of current knowledge engineering methodologies can take advantage of the acquired experience in these methodologies. In addition, both the existing tools and the developed ontology libraries and problem solving method libraries can be reused.

Although these methodologies are not as extendable as the object-oriented ones, they have been applied to several projects with success.

3.2 Aspects not addressed

Most of the problems subject to knowledge engineering methodologies are present in designing MAS: knowledge acquisition, modelling and reuse. Nevertheless, these methodologies conceive a knowledge based system as a centralised one. Thus, they do not address the distributed or social aspects of the agents, or their reflective and goal-oriented attitudes.

3.3 Existing solutions

Several solutions have been proposed for multi-agent systems modelling extending *CommonKADS* [42]. The main reason of the selection of this methodology among the knowledge engineering methodologies is that it can be seen as a European standard for knowledge modelling. *CommonKADS* defines the modelling activity as the building of a number of separate models that capture salient features of the system and its environment.

We will review the extensions *CoMoMAS* [16] and *MAS-CommonKADS* [19], though there have been other preliminary works [9, 25, 37, 47].

The *CoMoMAS* methodology

Glaser [16] proposes an extension to the methodology *CommonKADS* [42] for MAS modelling. The following models are defined:

- *Agent Model:* this is the main model of the methodology and define the agent architecture and the agent knowledge, that is classified as social, cooperative, control, cognitive and reactive knowledge.
- *Expertise Model:* describes the cognitive and reactive competences of the agent. It distinguishes between task, problem solving (PSM) and reactive knowledge. The *task knowledge* contains the task decomposition knowledge described in the task model. The *problem-solving knowledge* describes the problem solving methods and the strategies to select them. The *reactive knowledge* describes the procedures for responding to stimuli.
- *Task Model:* describes the task decomposition, and details if the task are solved by a user or an agent.
- *Cooperation Model:* describes the cooperation between the agents. using conflict resolution methods and cooperation knowledge (communication primitives, protocols ad interaction terminology).
- *System Model:* defines the organisational aspects of the agent society together with the architectural aspects of the agents.
- *Design Model:* collects the previous models in order to operationalisate them, together with the non-functional requirements.

The *MAS-CommonKADS* methodology

This methodology [19] extends the models defined in *CommonKADS*, adding techniques from object-oriented methodologies (OOSE, OMT) and from protocol engineering for describing the agent protocols (SDL [20] and MSC96 [40]).

The methodology starts with a *conceptualisation phase* that is an informal phase for collecting the user requirements and obtaining a first description of the system from the user point of view. For this purpose, the use cases technique from OOSE [40] is used, and the interactions of these use cases are formalised with MSC (*Message Sequence Charts*) [39]. The methodology defines the models described below for the analysis and the design of the system, that are developed following a risk-driven life cycle. For each model, the methodology defines the constituents (entities to be modelled) and the relationships between the constituents. The methodology defines a textual template for describing every constituent and a set of activities for building every model, based on the development state of every constituent (empty, identified, described or validated). These activities facilitate the management of the project.

This extension defines the following models:

- *Agent Model:* describes the main characteristics of the agents, including reasoning capabilities, skills (sensors/effectors), services, goals, etc. Several techniques are proposed for agent identification, such as analysis of the actors of the conceptualisation phase, syntactic analysis of the problem statement, application of heuristics for agent identification, reuse of components (agents) developed previously or usage of CRC cards, which have been adapted for agent oriented development.
- *Task Model:* describes the tasks (goals) carried out by agents, and task decomposition, using textual templates and diagrams.
- *Expertise Model:* describes the knowledge needed by the agents to carry out the tasks. The knowledge structure follows the KADS approach, and distinguishes domain, task, inference and problem solving knowledge. Several instances of this model are developed for modelling the inferences on the domain, on the agent itself and on the rest of agents. The authors propose the distinction between *autonomous problem solving methods*, that decompose a goal into subgoals that can be directly carried out by the agent itself and *cooperative problem solving methods*, that decompose a goal into subgoals that are carried out by the agent in cooperation with other agents.
- *Coordination Model:* describes the conversations between agents, that is, their interactions, protocols and required capabilities. The development of the model defines two milestones. The first milestone is intended to identify the conversations and the interactions. The second milestone is intended to improve these conversation with more flexible protocols such as negotiation and identification of groups and coalitions. The interactions are modelled using the formal description techniques MSC (Message Sequence Charts) and SDL (Specification and Description Language).
- *Organisation Model:* describes the organisation in which the MAS is going to be introduced and the organisation of the agent society. The description of the multiagent society uses an extension of the object model of OMT, and describes the agent hierarchy, the relationship between the agents and their environment, and the agent society structure.
- *Communication Model:* details the human-software agent interactions, and the human factors for developing these user interfaces.
- *Design model:* collects the previous models and is subdivided into three submodels: *application design:* composition or decomposition of the agents of the analysis, according to pragmatic criteria and selection of the most suitable agent architecture for each agent; *architecture design:* designing of the relevant aspects of the agent network: required network, knowledge and telematic facilities and *platform design:* selection of the agent development platform for each agent architecture.

This methodology has been successfully applied in several research projects in different fields, as intelligent network management (project CICYT TIC94-9139 *PROTEGER: Multi-Agent System for Network and Service Management*) and development of hybrid systems with multiagent systems (project ESPRIT-9119 MIX, Modular Integration of Symbolic and Connectionist Knowledge Based Systems).

4 Other approaches

The methodology *Cassiopeia*

The methodological approach called *Cassiopeia* [7], distinguishes three main steps for designing a MAS, applied to the robot soccer teams domain. Firstly, the elemental agent behaviours are listed using functional or object oriented techniques. Then the relational behaviours are analysed, that is, the dependencies between the agents are studied using a coupling graph. Finally, the dynamics of the organisation are described, that is, who can start or end a coalition, by analysing the coupling graph.

Cooperative Information Agents design

Verharen [46] proposes a methodology from a business process perspective. The methodology proposes the following models:

- *Authorisation model:* describes the authorised communication and obligations between the organisation and the environment and the internal communications using authorisation diagrams. After identifying the current situation, it is redesigned for improving the efficiency of the business processes.
- *Communication model:* refines the previous model describing in detail the contracts between the agents, using petri nets. The transactions between the agents are modelled using transaction diagrams that describe the relationship between speech-acts and goals.
- *Task model:* specifies the task decomposition using task diagrams.
- *Universe of Discourse model:* concerns the modelling of the content of the messages exchanged between the agents, using object-oriented techniques.

5 Formal Approaches

Several formal approaches have tried to bridge the gap between formal theories and implementations [10]. Formal agent theories are [10] *agent specifications* that allow the complete specification of the system. Though formal methods are not easily scalable in practice [13], there are specially useful for verifying and analysing critical applications, prototypes and complex cooperating systems.

Traditional formal languages such as Z have been used [30], providing an elegant framework for describing a system at different levels of abstractions. Since there is no notion of time in Z [30, 10, 13], it is less well suited to specifying agent interactions.

Another approach has been the use of temporal modal logics [49] that allows the representation of dynamic aspects of the agents and a basis for specifying, implementing and verifying agent based systems. The implementation of the specification can be done [49] by directly executing the agent specification with a language such as *Concurrent Metatem* [14] or by compiling the agent specification.

The usage of formal languages for multi-agent specification such as *DESIRE* [4] are a very interesting alternative to be used as detailed design language in any methodology. *DESIRE* (framework for DEsign and Specification of Interacting REasoning components) proposes a component-based perspective based on a task decomposition.

6 Techniques based on the experience of agent developers

The definition of methodologies for developing multiagent systems is a recent development. Nevertheless, multiagent systems have been successfully applied to different fields using different multiagent platforms. During this application, some agent developers have taken a software engineering perspective. Although they have not "formally" defined an AO methodology, they have given general guidance for MAS development. In this section, some of these recommendations are reviewed. Another important contribution is a collection of common mistakes of agent system developers [50].

The *ARCHON* experience

ARCHON [6, 45] is a complete development environment for MAS, which proposes a methodology for analysing and designing MAS.

The *analysis* combines a *top-down* approach, that identifies the system goals, the main tasks and their decomposition and a *bottom-up* approach, that allows the reuse of preexisting systems, constraining the top-down approach.

The *design* is subdivided into agent community design and agent design. The *agent community* design defines the agent granularity and the role of each agent. Then the authors propose the design of the user interfaces. Finally, the skills and interchanged messages are listed. The *agent design* encodes the skills for each agent (plans, rules, etc.).

The *MADE* experience

MADE [51, 36] is a development environment for rapid prototyping of MAS. It proposes a development methodology for designing MAS, extending the five stages for knowledge acquisition proposed by Buchanan *et al* [11]: Identification, Conceptualisation, Decomposition (added for agent identification), Formalisation, Implementation and Testing (here the integration of the MAS is added).

Coordination languages

There are several coordination languages that can be an alternative to interaction modelling and included in an AO methodology: (1) using *finite state representation* for conversations such as COOL [1] and *AgentTalk* [29]; (2) using a *formal language* which takes advantage of formal description techniques of protocol engineering such as *Yubarta* [38].

The *AWIC* Method

The *AWIC* method [34] proposes an iterative design approach. In every cycle, five models are developed, an agent is added to the system, and the overall system is tested. The proposed models are:

– *A: The agent model*. The developing of this model consists of the identification of the active agents of the problem, the specification of their tasks, sensors, actuators, world knowledge and planning abilities. Then a suitable agent architecture should be specified for each agent.
– *W: The world model*. This model represents the environment the agents operate in, detailing the world laws that minimise the coordination between agents and testing if the agent tasks are feasible in the world.
– *I: The interoperability model*. This model defines how the world reflects the actions of the agents and how the agents perceive the world.
– *C: The coordination model*. This models specify the protocols and interchanged messages among agents and study the suitability of joint planning or social structuring.

The decentralising refinement method

An interesting approach to bridge the gap among theory and practice is the decentralising refinement method [44]. This method proposes to start with a centralised solution to the problem. Then a general problem-solving method is abstracted out. The next step is the identification of the assumptions made on the agents' knowledge and capabilities, and the relaxation of these assumptions in order to obtain more realistic versions of the distributed system. Finally, the system is specified with a formal language. The method takes into account the reuse of the problem-solving methods, by identifying connections among parts of the problems and the agents that solve them.

7 Conclusions

This article has shown that there are several emerging agent-oriented methodologies. The reviewed methodologies are mainly extensions to known object-oriented or knowledge engineering methodologies. Which relevant aspects of object-oriented and knowledge engineering can be reused and which aspects are not covered have been discussed. In addition, the models and modelling process of these agent-oriented methodologies have been shown.

Several open issues not included in the reviewed methodologies can be cited, such as mobile agents design and user interface design [17] that is mentioned but not particularly developed in any methodology.

After the reviewing of these agent-oriented methodologies, several questions can arise:

– *Why are AO methodologies necessary?* The question of the need of AO methodologies have been mentioned previously in [22, 23, 33, 13]. Obviously, the engineering approach [13] to agent-based systems development is a key factor for their introduction in industry. This principled development will be specially needed as the number of agents in a system increases. The standard advantages of an engineering approach, such as management, testing and reuse should be applied in the development of agent-based systems.

– *Is agent technology mature enough for defining agent-oriented methodologies?* As long as there are no standard definitions of an agent, an agent architecture, or an agent language, we could think that the methodologies presented here will only be used by individual researchers to program their agent based application using their own agent language, architectures and theories. The methodologies reviewed here have shown that there is a conceptual level for analysing the agent-based systems, no matter the agent theory, agent architecture or agent language they are supported by. This conceptual level should describe:
    - *Agent Models:* the characteristics of each agent should be described, including skills (sensors and effectors), reasoning capabilities and tasks.
    - *Group/Society Models:* the relationships and interactions between the agents.

  The lack of standard agent architectures and agent programming languages is actually the main problem for operationalising the models, or providing useful "standard" code generation. Since there is no standard agent architecture, the design of the agents needs to be customised to each agent architecture. Nevertheless, the analysis models are independent of the agent architectures, they describe what the agent-based system has to do, but not how this is done.
– *What is the relationship between AO methodologies and agent architectures*
  Agent architectures are taken into account in different ways in two of the reviewed methodologies. *CoMoMAS* selects the agent architecture during the analysis, while *MAS-CommonKADS* considers that it is a design issue, and the agent architecture should be selected depending on the requirements of the analysis. In addition, *MAS-CommonKADS* proposes an expertise model of a generic agent architecture that guides the knowledge acquisition process.

  A first solution to the problem of the selection of an agent architecture is addressed by Müller [35], that presents some guidelines about which type of architecture is best suited for which type of application.
– *Are the reviewed AO methodologies just individual efforts, or are they converging efforts?*
  As we have stated previously, the reviewed AO methodologies can be compared since they use the same key concepts: mental state, tasks, interactions and group modelling. They propose complementary modelling techniques, though the degree of elaboration of these methodologies is quite different.

References

1. Mihai Barbuceanu and Mark S. Fox. Capturing and modeling coordination knowledge for multi-agent systems. *Journal on Intelligent and Cooperative Information Systems*, July 1996.
2. Alan H. Bond and Les Gasser. An analysis of problems and research in DAI. In Alan H. Bond and Les Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 3–36. Morgan Kaufmann Publishers: San Mateo, CA, 1988.
3. Grady Booch. *Object-Oriented Design with Applications*. Benjamin/Cummings, Redwood City, CA, 1991.

4. F. M. T. Brazier, B. M. Dunin-Keplicz, N. R. Jennings, and Treur J. DESIRE: Modelling multi-agent systems in a compositional formal framework. *Int Journal of Cooperative Information Systems*, 1(6):67–94, January 1997.
5. Birgit Burmeister. Models and methodology for agent-oriented analysis and design. In K Fischer, editor, *Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Systems*, 1996. DFKI Document D-96-06.
6. David Cockburn and Nick R. Jennings. ARCHON: A distributed artificial intelligence system for industrial applications. In G. M. P. O'Hare and N. R Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 319–344. John Wiley & Sons, 1996.
7. Anne Collinot, Alexis Drogoul, and Philippe Benhamou. Agent oriented design of a soccer robot team. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*, pages 41–47, Kyoto, Japan, December 1996.
8. Rational Software Corporation. *Unified Modelling Languaje (UML) version 1.0*. Rational Software Corporation, 1997.
9. Rose Dieng. Specifying a cooperative system through agent-based knowledge acquisition. In *Proceedings of the International Workshop on Cooperative Systems (COOP'95)*, pages 141–160, Juen-les-Pis, January 1995. Also published in *Proc. of the 9th International Workshop on Acquisition Knowledge for Knowledge-Based Systems*, Banff, Canada, February - March 1995, pages 20-1 – 20-20.
10. M. d'Inverno, M. Fisher, A. Lomuscio, M. Luck, M. de Rijke, M. Ryan, and M. Wooldridge. Formalisms for multi-agent systems. *The Knowledge Engineering Review*, 3(12), 1997.
11. B. G. Buchanan et al. Constructing an expert system. In F. Hayes-Roth, D. A. Waterman, and D. Lenat, editors, *Building Expert Systems*. Addison-Wesley, 1983.
12. FIPS Pub 183. Integration definition for function modeling (IDEF0). Software Standard. Modelling techniques. FIPS Pub 183, Computer Systems Laboratory National Institute of Standards and Technology, Gaithersburg, Md. 20899, 1993.
13. M. Fisher, J. Müller, M. Schroeder, G. Staniford, and G. Wagner. Methodological foundations for agent-based systems. In *Proceedings of the UK Special Interest Group on Foundations of Multi-Agent Systems (FOMAS). Published in Knowledge Engineering Review (12) 3, 1997*, 1997. http://www.dcs.warwick.ac.uk/ fomas/fomas96/abstracts/ker3.ps.
14. M. Fisher and M. Wooldridge. On the formal specification and verification of multi-agent systems. *International Journal of Cooperative Information Systems*, 1(6):37–65, 1997.
15. Les Gasser and Jean-Pierre Briot. Object-based concurrent processing and distributed artificial intelligence. In Nicholas M. Avouris and Les Gasser, editors, *Distributed Artificial Intelligence: Theory and Praxis*, pages 81–108. Kluwer Academic Publishers: Boston, MA, 1992.
16. Norbert Glaser. *Contribution to Knowledge Modelling in a Multi-Agent Framework (the CoMoMAS Approach)*. PhD thesis, L'Universtité Henri Poincaré, Nancy I, France, November 1996.
17. Lynne E. Hall. User design issues for distributed artificial intelligence. In G. M. P. O'Hare and N. R Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 543–556. John Wiley & Sons, 1996.
18. D. Harel. Statecharts: A visual formalism for complex systems. *Sci. Computer Program*, 8:231–247, 1987.
19. Carlos A. Iglesias, Mercedes Garijo, José C. González, and Juan R. Velasco. Analysis and design of multiagent systems using MAS-CommonKADS. In AAAI'97 Workshop on Agent Theories, Architectures and Languages, Providence, RI, July 1997. ATAL. (An extended version of this paper has been published in *INTELLIGENT AGENTS IV: Agent Theories, Architectures, and Languages*, Springer Verlag, 1998.

20. ITU-T. Z100 (1993). CCITT specification and description language (SDL). Technical report, ITU-T, June 1994.
21. I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object-Oriented Software Engineering. A Use Case Driven Approach*. ACM Press, 1992.
22. N. R. Jennings and A. J. Jackson. Agent-based meeting scheduling: A design and implementation. *Electronic Letters, The Institution of Electrical Engineering*, 31(5):350–352, March 1995.
23. N. R. Jennings and M. Wooldridge. Applying agent technology. *Applied Artificial Intelligence*, 9(6):357–370, 1995.
24. Elisabeth A. Kendall, Margaret T. Malkoun, and Chong Jiang. A methodology for developing agent based systems for enterprise integration. In D. Luckose and Zhang C., editors, *Proceedings of the First Australian Workshop on DAI*, Lecture Notes on Artificial Intelligence. Springer-Verlag: Heidelberg, Germany, 1996.
25. J. Kingston. Modelling interaction between a KBS and its users. *Newsletter of BCS SGES Methodologies Interest Group*, 1, August 1992. Also available from AIAI as AIAI-TR-141.
26. D. Kinny. The AGENTIS agent interaction model. In J. P. Müller, M. P. Singh, and A. S. Rao, editors, *Intelligent Agents V — Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg, 1999. In this volume.
27. David Kinny, Michael Georgeff, and Anand Rao. A methodology and modelling technique for systems of BDI agents. In W. van der Velde and J. Perram, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World MAAMAW'96, (LNAI Volume 1038)*. Springer-Verlag: Heidelberg, Germany, 1996.
28. K. Kosanke. *CIMOSA - A European Development for Enterprise Integration*. IOS Press, 1993.
29. Kazushiro Kuwabara, Toru Ishida, and Nobuyasu Osato. AgenTalk: Coordination protocol description for multiagent systems. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, page 455, San Francisco, CA, June 1995.
30. Michael Luck, Nathan Griffiths, and Mark d'Inverno. From agent theory to agent construction: A case study. In J. P. Müller, M. Wooldridge, and N. R. Jennings, editors, *Intelligent Agents III (LNAI 1193)*, Lecture Notes in Artificial Intelligence. Springer-Verlag: Heidelberg, Germany, 1997.
31. B. Moulin and L. Cloutier. Collaborative work based on multiagent architectures: A methodological perspective. In Fred Aminzadeh and Mohammad Jamshidi, editors, *Soft Computing: Fuzzy Logic, Neural Networks and Distributed Artificial Intelligence*, pages 261–296. Prentice-Hall, 1994.
32. Bernard Moulin and Mario Brassard. A scenario-based design method and an environment for the development of multiagent systems. In D. Lukose and C. Zhang, editors, *First Australian Workshop on Distributed Artificial Intelligentce, (LNAI volumen 1087)*, pages 216–231. Springer-Verlag: Heidelberg, Germany, 1996.
33. H. Jürgen Müller. (Multi)-agent systems engineering. In *Second Knowledge Engineering Forum*, Karlsruhe, February 1996.
34. H. Jürgen Müller. Towards agent systems engineering. *International Journal on Data and Knowledge Engineering. Special Issue on Distributed Expertise*, (23):217–245, 1996.
35. J. P. Müller. The right agent (architecture) to do the right thing. In J. P. Müller, M. P. Singh, and A. S. Rao, editors, *Intelligent Agents V — Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg, 1999. In this volume.

36. G.M.P O'Hare and M.J. Wooldridge. A software engineering perspective on multi-agent system design: Experience in the development of MADE. In Nicholas M. Avouris and Les Gasser, editors, *Distributed Artificial Intelligence: Theory and Praxis*, pages 109–127. Kluwer Academic Publishers: Boston, MA, 1992.
37. Arturo Ovalle and Catherine Garbay. Towards a method for multi-agent system design. In M. A. Bramer and R. W. Milne, editors, *Proceedings of Expert Systems 92, the 12th Annual Technical Conference of the British Computer Society Specialist group on Expert Systems, Research and Development in Expert Systems IX*, British Computer Society Conference Series, pages 93–106, Cambridge, U.K., December 1992. Cambridge University Press.
38. Alejandro Quintero, María Eugenia Ucrós, and Silvia Takhashi. Multi-agent systems protocol language specification. In *Proccedings of the CIKM Workshop on Intelligent Information Agents*, December 1995.
39. Björn Regnell, Michael Andersson, and Johan Bergstrand. A hierarchical use case model with graphical representation. In *Proceedings of ECBS'96, IEEE International Symposium and Workshop on Engineering of Computer-Based Systems*, March 1996.
40. Ekkart Rudolph, Jens Grabowski, and Peter Graubmann. Tutorial on message sequence charts (MSC). In *Proceedings of FORTE/PSTV'96 Conference*, October 1996.
41. J. Rumbaugh, M.Blaha, W. Premerlani, F. Eddy, and V. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, 1991.
42. A. Th. Schreiber, B. J. Wielinga, J. M. Akkermans, and W. Van de Velde. CommonKADS: A comprehensive methodology for KBS development. Deliverable DM1.2a KADS-II/M1/RR/UvA/70/1.1, University of Amsterdam, Netherlands Energy Research Foundation ECN and Free University of Brussels, 1994.
43. Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, March 1993.
44. Munindar P. Singh, Michael N. Huhns, and Larry M. Stephens. Declarative representations of multiagent systems. *IEEE Transactions on Knowledge and Data Engineering*, 5(5):721–739, October 1993.
45. L. Z. Varga, N. R. Jennings, and D. Cockburn. Integrating intelligent systems into a cooperating community for electricity distribution management. *International Journal of Expert Systems with Applications*, 7(4):563–579, 1994.
46. Egon M. Verharen. *A Language-Action Perspective on the Design of Cooperative Information Agents*. PhD thesis, Katholieke Universieit Brabant, the Netherlands, March 1997.
47. Hans-Peter Weih, Joachim Schue, and Jacques Calmet. CommonKADS and cooperating knowledge based systems. In *Proceedings of the 4th KADS User meeting*, GMD, Bonn, 1994.
48. R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software*. Prentice-Hall, 1990.
49. M. Wooldridge. Agents and software engineering. *AI*IA Notizie XI*, 3, September 1998.
50. M. Wooldridge and N. R. Jennings. Pitfalls of agent-oriented development. In P. Sycara and M. Wooldridge, editors, *Agents '98: Proceedings of the Second International Conference on Autonomous Agents*. ACM Press, May 1998.
51. Michael Wooldridge, Greg O'Hare, and Rebecca Elks. FELINE: A Case Study in the Design and Implementation of a Co-operating Expert System. In *Proceedings of the International Conference on Expert Systems and their Applications (Avignon-91*. Avignon-91, May 1991.