Analysis and Design of Multiagent Systems Using MAS-CommonKADS*

Carlos A. Iglesias , Mercedes Garijo ,
José C. González and Juan R. Velasco

Abstract. This article proposes an agent-oriented methodology called *MAS-CommonKADS* and develops a case study. This methodology extends the knowledge engineering methodology *CommonKADS* with techniques from object-oriented and protocol engineering methodologies. The methodology consists of the development of seven models: *Agent Model*, that describes the characteristics of each agent; *Task Model*, that describes the tasks that the agents carry out; *Expertise Model*, that describes the knowledge needed by the agents to achieve their goals; *Organisation Model*, that describes the structural relationships between agents (software agents and/or human agents); *Coordination Model*, that describes the dynamic relationships between software agents; *Communication Model*, that describes the dynamic relationships between human agents and their respective personal assistant software agents; and *Design Model*, that refines the previous models and determines the most suitable agent architecture for each agent, and the requirements of the agent network.

1 The MAS-CommonKADS methodology

MAS-CommonKADS [13] extends *CommonKADS* [28], for multiagent systems (MAS) modelling, adding techniques from object oriented (OO) methodologies such as *Object Modelling Technique* (OMT) [26], *Object Oriented Software Engineering* (OOSE) [15] and *Responsibility Driving Design* (RDD) [31] and from protocol engineering for describing the agent protocols, such as *Specification and Description Language* (SDL) [14] and *Message Sequence Charts* (MSC96) [25]). The methodology defines the following models:

- *Agent model (AM):* specifies the agent characteristics: reasoning capabilities, skills (sensors/effectors), services, agent groups and hierarchies (both modelled in the organisation model).
- *Task model (TM):* describes the tasks that the agents can carry out: goals, decompositions, ingredients and problem-solving methods, etc.
- *Expertise model (EM):* describes the knowledge needed by the agents to achieve their goals.
- *Organisation model (OM):* describes the organisation into which the MAS is going to be introduced and the social organisation of the agent society.
- *Coordination model (CoM):* describes the conversations between agents: their interactions, protocols and required capabilities.
- *Communication model (CM):* details the human-software agent interactions, and the human factors for developing these user interfaces.
- *Design model (DM):* collects the previous models and consists of three submodels: *network design* for designing the relevant aspects of the agent network infrastructure (required network, knowledge and telematic facilities); *agent design* for dividing or composing the agents of the analysis, according to pragmatic criteria and selecting the most suitable agent architecture for each agent; and *platform design* for selecting the agent development platform for each agent architecture.

The application of the methodology consists of the development of the different models. Each model consists of constituents (the entities to be modelled) and relationships between the constituents. A textual template is defined for each constituent in order to describe it. The states of the constituents describe their development: empty, identified, described or validated.

The software process model of the methodology combines the risk-driven approach with the component-based approach. The general process is risk driven, that is, in every cycle the states of the models to be reached are defined for reducing the perceived risks. When a state consists of identifying components, the developed components (agents, services, knowledge bases, etc.) are candidates for reusing.

In order to illustrate the application of the methodology, we will develop a case study, called *The Travel Agency*. The problem consists of building a system that is consulted by a user for booking a flight, and answers with the cheapest available flights with lowest probability of being delayed. The system will be run by any company, and the information of the flights will be available from the airlines.

2 Conceptualisation

During this phase we will carry out an elicitation task to obtain a preliminary description of the problem. This is carried out following a user-centered approach by determining some use cases (scenarios) which can help us to understand informal requirements and to test the system. Use cases are described using OOSE notation and the interactions are formalised with MSC (Message Sequence Charts) [25, 24].

We can identify one user role: *the traveller*, a person who wishes to travel. The following information should be supplied: departure date (*dd*), arrival date (*ad*) and

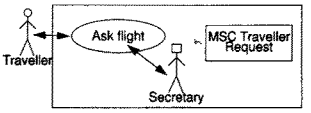

Fig. 1. Use case diagram

destination (*dest*). Two scenarios are identified: the system answers with an available flight (*num_flight*) or with no available flight (and the cause). If there is no available flight, the user can change the flight data. The interaction between the user and the system is represented using the use case notation of OOSE [15] (Fig. 1, notation extended as explained in 3.1). The interactions of the use cases are formalised using MSC as a notation (Fig. 2). In this figure two message interchange alternatives are combined with the alternative (*alt*) operator. A basic MSC contains the description of the asynchronous communication between entities called instances, and has primitives for local actions, timers (set, reset and time-out), process creation, process stop, coregions, and inline operators expressions for composition of event structures (alternative, parallel composition, iteration, exception and optional regions). The purpose in this phase is to get an idea of the interactions, but they will be refined later in the coordination model, specifying the data/knowledge interchanged and the speech-act of each interaction.

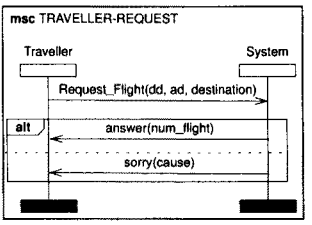

Fig. 2. MSC Traveller request use case diagram

3 Analysis

The results of this phase will be the requirements specification of the MAS through the development of the models previously described, except for the design model. These models are developed in a risk-driven way, and the steps are:

- *Agent modelling:* developing initial instances of the agent model for identifying and describing the agents.

- *Task modelling:* task decomposition and determination of the goals and ingredients of the tasks.
- *Coordination modelling:* developing the coordination model for describing the interactions and coordination protocols between the agents.
- *Knowledge modelling:* modelling of the knowledge on the domain, the agents (knowledge needed to carry out the tasks and their proactive behaviour) and the environment (beliefs and inferences of the world, including the rest of agents).
- *Organisation modelling:* developing the organisation model. Depending on the type of project, it may be necessary to model the organisation of the enterprise in which the MAS is going to be introduced for studying the feasibility of the proposed solution. In this case, two instances of the organisation model are developed: before and after the introduction of the MAS. This model is also used to model the software agent organisation. Another approach to define a social level for MAS extending *CommonKADS* is presented in [11].

3.1 Agent Modelling

Agents can be identified with the following strategies (or a combination of them):

- Analysis of the actors of the use cases defined in the conceptualisation phase. The actors of the use cases delimit the external agents of the system. Several similar roles (actors) can be mapped onto one agent to simplify the communication.
- Analysis of the statement of the problem. The syntactic analysis of the problem statement can help to identify some agents. The candidate agents are the subjects of the sentences, the active objects. The actions carried out by these subjects should be developed by the agents as goals (with initiative) or services (under demand).
- Usage of heuristics. The agents can be identified determining whether there is some *conceptual distance* [3]: knowledge distribution, geographical distribution, logical distribution or organisational distribution.
- An initial task and expertise models can help us to identify the necessary functions and the required knowledge capabilities, resulting in a preliminary definition of the agents. The goals of the tasks will be assigned to the agents.
- Application of the *internal use cases* technique. This technique is based on RDD [31] and its CRC (Class Responsibility Collaboration) cards. Taking as input the use cases of the conceptualisation phase and some initial agents, we can think that each agent "uses" other agent(s), and can use these agents with different roles. The use case notation (Fig. 1 and 3) is extended for showing human agents (with the round head) and software agents (with the squared head). When an agent needs to use an agent for a particular function (for example, evaluate something), we look for such an agent in our agent-library for reusing, combining in this way the top-down and bottom-up approach.
- Application of the enhanced CRC cards. A CRC is filled for each agent, describing its class. Each CRC is divided into five columns: goals assigned, plans for achieving these goals, knowledge needed to carry out the plans, collaborators in these plans, and services used in the collaboration. The back side of the CRC is used for annotations or extended description of the front side.

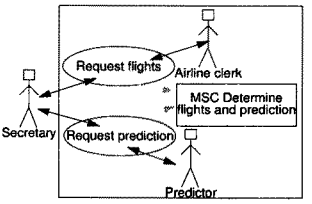

Fig. 3. Internal use case diagram

In the proposed case study, we identify:

- Since there is a user (human agent), as a general rule, we create a user-interface agent derived from an interface agent class for each human agent. In this case, it will be called *Secretary*. The type of interaction (menu-based, etc.) between a human agent and his/her agent assistant should be modelled in the communication model.
- Now we can recognise a knowledge distance (we need an expert in predicting flights without delays), with the role (class) of *Predictor*.
- There is also a geographical distance, the information of the available airlines can only be accessed through *Airlines-Clerk* agents. This information will be requested from *Airlines-Clerk* agents, so it can be useful to define a group for these agents and send multicast messages. This group will be modelled in the organisation model.

We should then fill the textual template of the agent model for each identified agent, that includes its name, type, role, position, a description, offered services, goals, skills (sensors and effectors), reasoning capabilities, general capabilities norms, preferences and permissions.

The approach followed here is quite different from the approach of agent identification in synthetic ecosystems [23], since we suppose that agents will be rather complex (because of their architecture) and we will try to limit the number of agents.

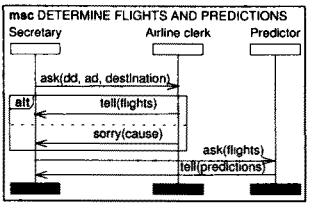

Fig. 4. MSC internal use case diagram

3.2 Task Modelling

Tasks are decomposed following a top-down approach, and described in an and/or tree. The description of a task [8] includes its name, a short description, input and output ingredients, task structure, its control, frequency of application, preconditions and required capabilities of the performers.

The potential benefits of the development of this model are the documentation of the activities of the organisation before and after the introduction of the multiagent system. This documentation serves for supporting the maintenance and management of changes in the organisation and for supporting project feasibility assessment.

3.3 Coordination Modelling

The coordination model has two milestones: (1) definition of the communication channels and building of a prototype; (2) analysis of the interactions and determination of complex interactions (with coordination protocols).

The first phase consists of the following steps:

1. Describe the prototypical scenarios between agents using MSC notation (Fig. 4). The conversations are identified taking as an input the results of the techniques used for identifying agents. During this first stage, we will consider that every conversation consists of just one single interaction and the possible answer.
2. Represent the events (interchanged messages) between agents in event flow diagrams (also called service charts) (Fig. 5). These diagrams collect the relationships between the agents via services.
3. Model the data interchanged in each interaction. The expertise model can help us to define the interchanged knowledge structures. These interchanged data are shown in the event flow diagram between squared brackets.
4. Model each interaction with the state transition diagrams of SDL (Specification and Description Language) [14] specifying speech-acts as inputs/outputs of message events (Fig. 6). These diagrams can be validated with the MSC diagrams.
5. Each state can be further refined in the task or expertise model.
6. Analyse each interaction and determine its synchronisation type: synchronous, asynchronous or future.

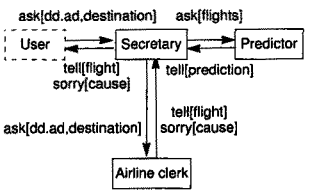

Fig. 5. Event flow diagram

The second phase consists of analysing the interactions for getting more flexibility (relaxing for example the user requirements), taking advantage of the parallelism [7], duplicating tasks using different methods or resolving detected conflicts. When a cooperation protocol is needed, we should consult the library of cooperation protocols and reuse a protocol definition. If there is no protocol suitable for our needs, it is necessary to define a new one. We can use HMSC (High level Message Sequence Charts) [14], which are very useful for this purpose. These diagrams (Fig. 7) show the road map (phases) of the protocol, and how the different phases (specified with MSC) are combined. A phase can be a simple MSC or another HMSC (e.g. *counterp*). The processing of the interactions is described using SDL state diagrams, and it is also necessary to fill in the textual protocol template specifying the required reasoning capabilities of the participants in the protocol. These capabilities can be described using one or several instances of the expertise model. The state diagrams consider three kinds of events: *message events*, events from other agents using message-passing; *external events*, events from the environment perceived through the sensors; and *internal events*, events that arise in an agent because of its proactive attitude.

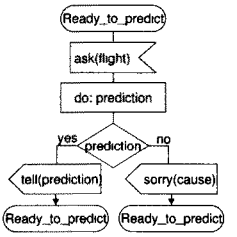

Fig. 6. SDL state diagram

The potential benefits of the development of this model are:

- The development of the coordination model is a means for specifying the prototypical interactions between the agents working on the resolution of a problem, together with the interactions with the environment. This model is used to store the decisions of the structure of communications and the protocols associated with these communications. The usage of these descriptions is twofold: the designer can reuse protocols and scenarios and the intelligent agent can select them at run time.
- MSC and SDL are formal description techniques with a well-defined syntax and semantics. The usage of these languages for specifying interactions in multiagent systems have been achieved by: (1) defining one signal type for each possible speech-act (message type); (2) associating a logical expression to each state name (using commentaries); and (3) considering internal events (similar to spontaneous transitions) for changes in the mental state of the agent motivated because of its proactive attitude. In addition, a multicast message has been proposed and requested from the MSC standardisation working group, for simplifying the specification of group

protocols. These languages have been used for supporting the system specification, design, documentation and definition of test cases.
- The development of this model can help in the maintenance and testing of a multiagent system.

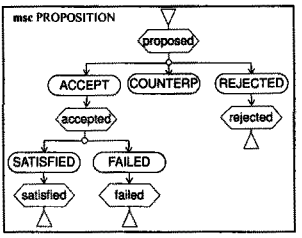

Fig. 7. HMSC diagram

3.4 Knowledge Modelling

The expertise model is used for modelling the reasoning capabilities of the agents to carry out their tasks and achieve their goals. Normally, several instances of the expertise model should be developed: modelling inferences on the domain (i.e. how to predict delays in flights, taxonomies of delays and flights, etc.); modelling the reasoning of the agent (i.e. problem solving methods to achieve a task, character of the agent, etc.) and modelling the inferences of the environment (how an agent can interpret the event it receives from other agents or from the world). When we have to develop the reasoning capabilities of an agent, we will reuse previously developed instances of the expertise model and adapt these instances to the new characteristics of the problem.

The expertise model [30][3] consists of the development of the *application knowledge* (consisting of *domain knowledge*, *inference knowledge* and *task knowledge*) and *problem solving knowledge*.

The usage of this model can take advantage of the work previously developed, for example for developing a planner [2].

Domain Knowledge: represents the declarative knowledge of the problem, modelled as concepts, properties, expressions and relationships using the *Conceptual Modelling Language* (CML) [27] or the graphical notation of the Object Model of OMT.

In our problem, if we are focusing just on the domain, we could identify concepts such as flight, airlines, delay, etc.; properties such as num_flight, ao, dd,.... These concepts are arranged in *domain models* that describe a particular relationship between

[3] A very practical approach to the development of this model can be found in [17].

themselves. For example, we could develop a causal model of what events cause a delay; a hierarchy of events, delays, etc. The road map of the developed domain models and their relationships are presented in *model schematas*.

Inference Knowledge: represents the inference steps performed for solving a task. There is a library of generic inference structures selected by the task type (diagnosis, assessment, etc.). These generic inference structures should be adapted to the problem. Consulting the library for modelling how to predict whether a flight is going to be delayed, we see that the task *Prediction* is not suitable, because this is used for suggesting what will happen next to the system. We found that the most suitable task is *Assessment*, whose inference structure (supposing no available norm) is shown in Fig. 8. The inference structure is a compound of predefined inference types (how the domain concepts can be used to make inferences, represented as ellipses) and domain roles (rectangles). This generic inference structure should be adapted to our problem.

After defining the inference structure, it is instantiated into a similar diagram for the domain.

Task Knowledge: represents the order of the inference structures. The notation consists of inference structures and task-method inference decomposition structures.

Problem Solving Method: during the design we should specify a Problem Solving Method (PSM) for each inference type: how the inference is carried out. The PSMs are arranged in libraries for reuse.

The potential benefits of the development of this model are the utilisation of a well-known knowledge level modelling framework, which has been successfully applied in several projects, and the provision of a library of generic components, specification languages and software tools.

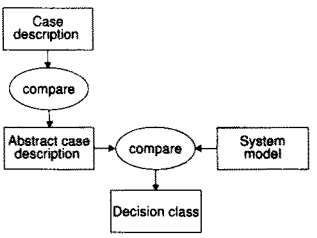

Fig. 8. Inference structure diagram

3.5 Organisation Modelling

CommonKADS defines the organisation model for modelling the organisation in which the knowledge based system is going to be introduced. Here the model is extended in the same way as the agent model for modelling the organisation of agents. This model shows the static or structural relationships between the agents, while the coordination model shows the dynamic relationships. The graphical notation of these models is based on the notation of the Object Model of OMT, adding a special symbol for distinguishing between agents and objects. An example of agent hierarchy diagram is shown in Fig. 9. The aggregation symbol is used for expressing agent groups.

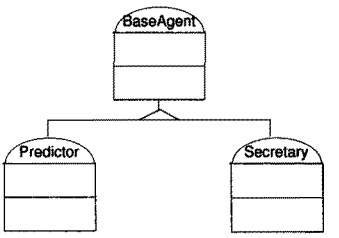

Fig. 9. Class agent diagram

The agent symbol is quite similar to the class symbol proposed in OMT, but has a different meaning. The upper box does not store the defined attributes as in OMT but the mental state and internal attributes of an agent, such as their goals, beliefs, plans, etc. The lower box stores the external attributes of the agents: services, sensors and effectors.

The inheritance relationship between agents is defined as the union of the values of the precedent classes for each attribute. For example, an agent class has its goals and the goals of the precedent agent classes. If an agent defines an attribute as exclusive, the values are overwritten.

The potential benefits of the development of this model is the specification of the structural relationships between human and/or software agents, and the relationship with the environment. The study of the organisation is a tool for the identification of possible impacts of the multiagent system when installed. In the same way, this model can provide information about the functions, workflow, process and structure of the organisation that allows the study of the feasibility of the proposed solutions. This model represents both class agent diagrams and instance agent diagrams, showing the particular relationships with the environment. In contrast with other paradigm (i.e. object oriented), the agent instance diagrams are frequently more relevant than the class agent diagrams.

4 Design

As a result of the analysis phase, an initial set of agents has been determined. During the design phase the design model is developed. This phase is extended for MAS and consists of [13]:

- *Agent network design:* the infrastructure of the MAS-system (so-called *network model* [12]) is determined, and consists of network, knowledge and coordination facilities. The agents (so-called *network agents*) that maintain this infrastructure are also defined, depending on the required facilities. Some of these required facilities can be:
  - *Network facilities:* agent name service, yellow/white pages service, de/registering and subscription service, security level, encryption and authentication, transport/application protocol, accounting service, etc.
  - *Knowledge facilities:* ontology servers, PSM servers, knowledge representation language translators, etc.
  - *Coordination facilities:* available coordination protocols and primitives, protocol servers, group management facilities, facilities for assistance in coordination of shared goals, police agents for detecting misbehaviours and the control of the usage of common resources, etc.

  The result of the common facilities shared by the agents allow the efficient communication between the agents and is expressed in an ontology, in the same way as the service ontology as defined by Nodine [22].
- *Agent design:* the most suitable architecture is determined for each agent, and some agents can be introduced or subdivided according to pragmatic criteria. Each agent is subdivided in modules for user-communication (from communication model), agent communication (from coordination model), deliberation and reaction (from expertise, agent and organisation models), external skills and services (from agent, expertise and task models). The agent design maps the functions defined in these modules onto the selected agent architecture.

  The issue of designing an agent architecture [5] is not addressed in the methodology, since the agent architecture is provided by the agent development platform.
- *Platform design:* selection of the software (multiagent development environment) and hardware that is needed (or available) for the system.

The potential benefits of the development of this model are:

- The decisions on the selection of a multiagent platform and an agent architecture for each agent are documented.
- The design model collects the information of the previously developed models and details how these requirements can be achieved.
- The design model for multiagent systems determines the common resources and needs of the agents and designs a common infrastructure managed by network agents. This facilitates modularity in the design.

5 Related Work

There are several proposals for defining an agent-oriented (AO) methodology.

Here we include a review and the relationship between these approaches and *MAS-CommonKADS*.

Kinny [18] defines a methodology for MAS extending OMT. He proposes two main levels: an external view for modelling the agent relationships (our organisation model) and the interactions (our coordination model). The internal view describes the mental state of a BDI (Belief-Desire-Intention) agent (our expertise and agent models). The modelling of the interactions is elaborated more in our coordination model. The internal view could be an interesting alternative to the our expertise model, though the expertise model offers a very elaborate framework for knowledge modelling.

Burmeister [6] describes an AO methodology, extending OO techniques. Three models are distinguished: agent model (our agent and expertise models), organisational model (our organisation model) and a cooperation model (our coordination model). She proposes a very interesting extension to the CRC cards and a clear development process. Our graphical notation for modelling interactions seems to be more detailed, and the knowledge modelling is elaborated more in the *CommonKADS* framework.

Kendall [16] proposes another AO methodology based on OO and enterprise modelling techniques. The use case model is very similar to our internal use cases. Coordination and knowledge modelling are not so well developed and the process development is not very clear.

MASB [20, 21] proposes an AO methodology that covers analysis and design. The *behaviour diagrams* are similar to the internal use case diagrams, and it proposes a new graphical notation (perhaps too complex) for modelling the agents. The conversation modelling is only mentioned.

CoMoMAS [10] proposes also an extension to *CommonKADS* for MAS. It has a very interesting extension to CML for MAS and a good redefinition of the expertise model. It also defines a new model for cooperation, but is less developed than our coordination model. Our model also proposes different graphical notations instead of just textual templates.

DESIRE [4] is a formal framework for multiagent modelling, that covers mainly our task, agent and expertise models. It could be suitable for specifying the design after the analysis phase.

CoLa [29] is a specification language for task decomposition, transactions and contracts, which are specified in our methodology in the task and coordination models. Our graphical notation could be easily mapped onto this specification language.

COOL [1] and *AgentTalk* [19] are an alternative to our coordination model. Our model takes advantage of the properties of formal description techniques and their standardised textual and graphical notation and semantics.

6 Conclusions and Future Work

The engineering approach [9] to agent-based systems development is a key factor for their introduction into the industry. This principled development will be specially needed

as long as the number of agents of the systems increase. The standard advantages of an engineering approach, such as management, testing and reutilisation should be applied in the development of agent-based systems.

This paper presents an agent-oriented methodology that covers the software development life cycle of a multiagent system, through the development of seven models, that can be reused. The software process model combines a risk-driven approach with a component-based approach.

This methodology integrates techniques from a well-known knowledge engineering methodology, *CommonKADS*, with techniques from object-oriented methodologies and protocol engineering. The application of techniques based on well-known techniques is intended to facilitate the learning of the methodology and to provide confidence to the managers with techniques that have been successfully applied.

For each model of the methodology, we have shown the standard development process and the graphical notation. The methodology also defines textual templates for each model, not included here, and some non-standard development processes.

This approach is currently being employed in real applications. The feedback from these applications will help to refine the methodology.

Our main effort has been the development of the new model, the Coordination Model. The rest of the models are subject of further improvement.

Our future work is focused on the development of a workbench for the methodology, since there is no integrated environment available.

Acknowledgements

We would like to thank Amalio F. Nieto, Mark Hallett and two anonymous referees for many suggestions concerning the content and presentation of this paper.

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. V. R. Benjamins, Leliane Nunes de Barros, and Valente Andre. Constructing planners through problem-solving methods. In B. Gaines and M. Musen, editors, *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, volume 1, pages 14–1/20, Banff, Canada, November 1996. KAW.
3. 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. Springer-Verlag: Heidelberg, Germany, 1988.
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):To appear, January 1997.
5. Joanna Bryson. Agent architecture as object oriented design. (In this volume).
6. 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.

7. E. H. Durfee, V. R. Lesser, and D. D. Corkill. Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, 1(1), March 1989.
8. Cuno Duursma, Olle Olsson, and Sundin Ulf. Task model defintion and task analysis process. Technical Report Technical report KADS-II/M5/VUB/TR/004/2.0 ESPRIT Project P5248, Free University Brussels and Swedish Institute of Computer Science, 1994.
9. M. Fisher, J. Müller, M. Schroeder, G. Staniford, and G. Wagne. 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.* http://www.dcs.warwick.ac.uk/ fomas/fomas96/abstracts/ker3.ps.
10. 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.
11. Rune E. Gustavsson. Multi agent systems as open societies - a design framework -. (In this volume).
12. C. A. Iglesias, J. C. González, and J. R. Velasco. MIX: A general purpose multiagent architecture. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI 1037)*, pages 251–266. Springer-Verlag: Heidelberg, Germany, 1996.
13. Carlos A. Iglesias, Mercedes Garijo, José C. González, and Juan R. Velasco. A methodological proposal for multiagent systems development extending CommonKADS. In B. Gaines and M. Musen, editors, *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, volume 1, pages 25–1/17, Banff, Canada, November 1996. KAW. Track Agent-Oriented Approaches To Knowledge Engineering.
14. ITU-T. Z100 (1993). CCITT specification and description language (sdl). Technical report, ITU-T, June 1994.
15. I. Jacobson, M. Christerson, P. Jonsson, and Övergaard. *Object-Oriented Software Engineering. A Use Case Driven Approach.* ACM Press, 1992.
16. 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.
17. John Kingston. Building a KBS for health and safety assessment. In *Applications and Innovations in Expert Systems IV, Proceedings of BCS Expert Systems '96*, pages 16–18, Cambridge, December 1996. SBES Publications. Also published as technical report: AIAI-TR-202, Artificial Intelligence Applications Institute, University of Edinburgh.
18. 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.
19. 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.
20. 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.
21. 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.

22. Marian H. Nodine and Amy Unruh. Facilitating open communication in agent systems: the infosleuth infrastructure. (In this volume).
23. Van Parunak, John Sauter, and Steve Clark. Toward the specification and design of industrial synthetic ecosystems. (In this volume).
24. 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.
25. Ekkart Rudolph, Jens Grabowski, and Peter Graubmann. Tutorial on message sequence charts (MSC). In *Proceedings of FORTE/PSTV'96 Conference*, October 1996.
26. J. Rumbaugh, M.Blaha, W. Premerlani, and V.Lorensen F. Eddy. *Object-Oriented Modeling and Design*. Prentice-Hall, 1991.
27. A. Th. Schreiber, B. J. Wielinga, and J. M. Akkermans W. Van de Velde. CML: The CommonKADS conceptual modelling language. Research report KADS-II/M2/RR/UvA/69/1.0, University of Amsterdam, Netherlands Energy Research Foundation ECN and Free University of Brussels, March 1994. Accepted for EKAW'94.
28. A. Th. Schreiber, B. J. Wielinga, and J. M. Akkermans 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.
29. Egon Verharen, Frank Dignum, and Sander Bos. Implementation of a cooperative agent architecture based on the language-action perspective. (In this volume).
30. B. J. Wielinga, W. van de Velde, A. Th. Schreiber, and H. Akkermans. Expertise model definition document. deliverable DM.2a, ESPRIT Project P-5248 /KADS-II/M2/UvA/026/1.1, University of Amsterdam, Free University of Brussels and Netherlands Energy Research Centre ECN, May 1993.
31. R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software*. Prentice-Hall, 1990.