

# Designing Methodologies Using System Levels

Woolaganathan Pillay and Johnson Kinyua

Department of Computer Science, University of Kwa-Zulu Natal, ZA,  
pillayw4@ukzn.ac.za; kinyuaj@ukzn.ac.za

**Abstract.** The need for methodologies to aid the development of high quality agent systems has been widely recognised and several methodologies have been proposed to fill this need. These methodologies differ widely in terms of the concepts and advocated processes and models. There have been a few attempts to unify some of these methodologies. This paper is a contribution to this effort. It proposes the use of system levels as a structuring mechanism for AOSE methodologies.

The framework presented generalises the knowledge level of Newell and the social level of Jennings and Campos and shows how these levels may be used to characterise the inter-agent level and intra-agent level commonly used in MAS methodologies. An agent-oriented methodology can be defined as a set of modelling techniques and methods that produce an instantiation of the inter-agent and intra-agent levels. We demonstrate the use of this framework by describing a task-driven methodology.

## 1 Introduction

One of the earliest methodologies for multiagent systems was proposed by Burmeister [3]. Since then there have been several attempts including AaLaadin [6], MAS-CommonKADS [10], CoCoMAS [9], MaSE [25], PASSI [4], GAIA [26]; [27], MESSAGE/UML [8], Prometheus [19], ROADMAP [12], Methodology for BDI Agents [15], Agent-Object-Relationship Modelling [22] [23], MASSIVE [16], and Tropos [5]. See [24] for a recent survey of AOSE methodologies and the field of AOSE in general. The methodologies differ substantially in terms of life-cycle coverage, concepts modelled, and modelling techniques and notations used.

There have been a few attempts to unify MAS methodologies. For example [14] and [13] propose the use of feature extraction from methodologies. They demonstrate that new methodologies can be assembled from these features.

This paper demonstrates that the inter-agent and intra-agent levels may usefully be used structure MAS methodologies. The rest of the paper is organised as follows. In section 2 we present an overview of the system levels for MAS. In section 3 we describe the inter-agent level models and section 4 describes the intra-agent models. Section 5 gives a comparison of methodologies presented here and other AOSE methodologies.

## 2 System levels for MAS

One of the most important activities in software engineering is the decomposition or modularisation of the system and we suggest that a MAS be decomposed into levels as described below. Developing a system as a series of levels has the advantage of a clear separation of concerns with the system being modelled at different, nearly independent levels.

The standard levels include the device level, the circuit level, the logic level, the register-transfer level, and the program level (or symbolic level) [21].

Each level is characterised by [17]:

- a *medium* that is to be processed e.g. symbols and expressions at the program level and bit vectors at the register-transfer level.
- *components* that provide primitive processing e.g. registers at the register-transfer level and memories and operations at the program level.
- *laws of composition* that allow the assembly of components into systems e.g. transfer paths at the register-transfer level and the notion of a program at the program level. These laws define a set of ways to combine the components.
- *laws of behaviour* that determine how the behaviour of the system depends on component behaviour and system structure e.g. logical operations at the register-transfer level and the way a program changes its memory during the course of execution at the program level. By behaviour is meant the system's change in states through time. Behaviour of the system is determined by components and their combinations.

Two new levels above the traditional levels have been suggested to cater for agents and multiagent systems. Table 1 summarises the upper levels of a multiagent system and is a generalisation of Newell's knowledge level and Jennings and Campos's [11] social level.

**Table 1.** Higher Systems Levels in MAS

LEVELS	SYSTEM	MEDIUM	COMPONENTS	COMPOSITION LAWS	BEHAVIOUR LAWS
Inter-agent Level	multiagent system	interactions	environments and agents	conversations	
Intra-agent Level	agent	percepts and actions	sensors and effectors	architecture	
Program Level	computers	symbols, expressions	memories, operations	designation and association	sequential interpretation

The inter-agent level describes the structure and behaviour of an MAS while the intra-agent level describes the structure and behaviour of an individual agent that may or may not be part of an MAS.

The *system* at the inter-agent level is an MAS. It is an interaction processing system composed of agents and environments. The agents' internal structure is not visible (they are black boxes) and they interact with each other and the environment. The term *conversation* is used in the broad sense to mean simply any sequence of interactions (among agents or between agents and the environment).

The system at the intra-agent level is the agent that continually processes its percepts (received through its sensors) and sets of actions in order to act on the environment through its effectors. The environment at this level is external to the agent and is therefore not part of the intra-agent level. The agent's internal structure or *architecture* defines the relationships between its sensors, actions, and effectors.

A general characteristic of all system levels is that there are many *instantiations* of each level. An instantiation is particular set of components, composition laws, medium, and unique laws of behaviour. An instantiation may describe a class of systems or a particular system.

An agent-oriented methodology can be defined as *a set of modelling techniques and methods that produce an instantiation of the inter-agent and intra-agent levels*. An instantiation is a description of the medium, components, composition laws, and behaviour laws at each level. We do not prescribe the order in which these levels should be instantiated.

The value of the framework for AOSE is its separation into the two levels and it makes clear the concepts that must be described at each level. It can thus be used to structure a methodology's models and methods.

## 2.1 The Intra-agent level

In terms of the framework presented above, we do not view the knowledge level as a separate level but as an instantiation of the intra-agent level. It describes a class of systems by providing particular instantiations of the agent's percepts and actions (i.e. knowledge), and unique behaviour laws.

The framework also does not describe any behaviour laws as this is dependent on particular approaches to developing multiagent systems.

A knowledge level characterisation of an agent separates an agent's reasoning mechanism from the representation it uses. Designing such an agent requires one to consider the agent's goals, knowledge (of the environment, other agents and its actions), a representation scheme for the knowledge, and a mechanism to process the representation.

An alternative approach to building individual agents is the behaviour-based approach. Here, reasoning and representation are not considered. Rather, the agent is designed bottom-up and consists of a hierarchy of increasingly more complex task achieving behaviours or competencies. The behaviour of the agent as a whole *emerges* from the interaction between the various behaviours.

## 2.2 The Inter-agent Level

At this level, the agents are viewed as black boxes and the engineer is concerned only with deciding how their interactions with each other and their environments can bring about system-wide behaviour that meets the requirements of the application.

One instantiation of this level is the *social level* introduced in [11] as a level above the knowledge level. This was felt necessary since the knowledge level cannot give an account of multiple entities and their interactions. The authors propose that the behaviour law at this level is the *principle of social rationality*—*“If a member of a responsible society can perform an action whose joint benefit is greater than its joint loss, then it may select that action”*.

In the framework described above, we do not require that the agent be able to reason about joint benefits or even be aware that it is part of a multiagent society. The more general approach will allow for agents to interact without communication or reasoning as done through the mechanism of *stigmergy* [7]. The principle of social rationality is seen as describing one instantiation of the inter-agent level.

## 2.3 The Model Suite

In the rest of the paper we show how these levels may be used to structure the phases and artifacts of a methodology by describing a task-driven methodology for engineering multiagent systems.

The methodology’s model suite is shown in Fig. 1. The first phase of the

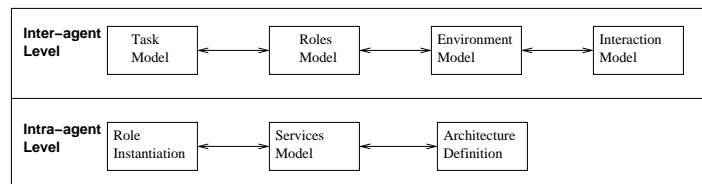


Fig. 1. Model Suite

methodology instantiates the inter-agent level and revolves around the concept of *tasks*, which are purposeful units of activity carried out by some entity in the domain. Tasks are used to identify *roles* that populate an *organisation*.

The second phase instantiates the intra-agent level by providing a specification for each agent that plays a role in the organisation. We illustrate the methodology by using a case study of a bookstore [1] described briefly below.

## 2.4 The Case Study

The Juul Møller Bokhandel (JMB) is a bookstore whose main clients are students of the Norwegian School of Management (NSM). JMB has an agreement with the NSM whereby JMB is granted early access to literature specifications for NSM courses.

Faculty members of NSM make literature specifications a few months before the year starts. For each course some books are required and some recommended. However, literature specifications are often late and unreliable. As a result books are often not available when required.

The ordering of books is the most demanding aspect of the business. Typically, orders are placed to distribution centres either by fax or email. Delivery times vary depending on whether the distributor has stock or the books have to be ordered direct from the publisher. If JMB is unable to source the books, it arranges for the book to be photocopied.

JMB uses regular office software, some industry-specific databases and a proprietary inventory and order system. The task is to provide software to improve the quality of service. In particular, the software should automate the selling and ordering of books.

## 3 Inter-agent Level Models

Recall that at the inter-agent level, the system is a society of agents and the key abstraction or metaphor at this level is that of an organisation. Thus, the purpose of the first phase is to produce an organisation design.

### 3.1 Organisation Design

The output of our organisation design phase is a *roles model* that documents the roles in MAS.

While the use of roles in AOSE methodologies is common, there is no agreement on how to identify roles. This is a crucial step in any methodology as the identification of agents usually follows in a straightforward manner from the roles (agents are assigned to play roles). So, role identification (and thus agent identification) has the most bearing on the nature and types of agents that eventually populate the MAS.

We propose *tasks* as the central concept in the early phases and describe a task-driven organisation design process. We view tasks as goal-oriented, purposeful activities. The organisation design can be done via the following steps.

- *Task Model*. Use UML activity diagrams and task schemata to discover, understand and document the tasks carried out in the domain. Re-factor the activity diagrams to create more roles and combine roles.
- *Roles Model*. Describe the roles using role schemas. Roles are characterised by responsibilities, rights (duties), and permissions.
- *Environments Model*. Model the environments within which roles are embedded.

### 3.2 Task Model

The first step is perhaps the most important in any methodology as it defines what follows (or can follow). Our methodology advocates a focus on tasks as a first step. We borrow heavily from the CommonKADS task model but the purpose of the model is different.

Tasks can be considered at different granularities. At the analysis level we consider a task at a coarse grained level as “purposeful (i.e. goal directed) activity carried out by an agent (human or artificial)” [20]. A task has the following attributes:

- *Goal*. Why the activities are carried out.
- *Knowledge*. The knowledge resources required and provided.
- *Inputs and Outputs*. The knowledge and other resources consumed and/or provided by the activity.

This phase is concerned mainly with process engineering. The analyst proceeds by first understanding and documenting current business processes. This understanding is iteratively built up from modelling the tasks carried out in the organisation. UML activity diagrams are useful to document tasks; they show a functional decomposition of business processes and include information about interactions between various sub-processes (see Fig. 2). In this initial exploration, the activity diagram *swim-lanes* represent people or existing computer systems. The next step focuses on *role identification*. Roles encapsulate an entity’s function and are identified by grouping “related” tasks. Roles are identified by a process we call *task refactoring* and involves postulating roles that improve the business processes and provide new opportunities for automation. The analyst, together with users and other stakeholders, reallocates tasks and defines new tasks for specific roles within the organisation.

Thus, role identification is done by grouping tasks. A consideration of *goals* is useful here. The analyst is guided by the following principles: tasks with a high degree of interaction between them should be grouped together (*task cohesion*); there should be a low degree of interaction (or dependencies) between roles (*low role coupling*); and roles should have a few clearly defined goals (*goal coherence*).

Role identification results in activity diagrams documenting the tasks that each role is responsible for (see Fig. 3). The swim-lanes now represent roles. A more detailed understanding of important tasks is now required. This is done by documenting the inputs and outputs of each task (including the knowledge and other resources required and provided). The task attributes can be documented by using a task schema as shown in Fig 4.

### 3.3 Roles Model

The roles identified in the previous step can now be documented. A role has the following attributes (adapted from the GAIA methodology [27]):

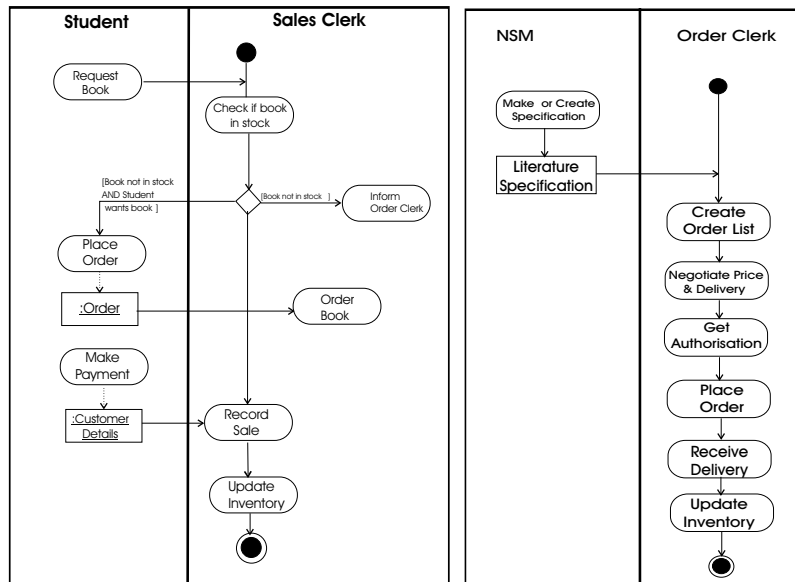


Fig. 2. Business Processes for Bookstore Case Study

- *Goals*. Goals provide the mechanism to unify tasks into a role. Goals may be operational (characterised by the outputs produced) or they may have quality attributes that describe performance measures.
- *Tasks*. The tasks assigned to this role.
- *Permissions*. Similar to GAIA in that it describes the resources that may be consumed by the role. A role has permission to either *read*, *change*, or *generate* a resource.

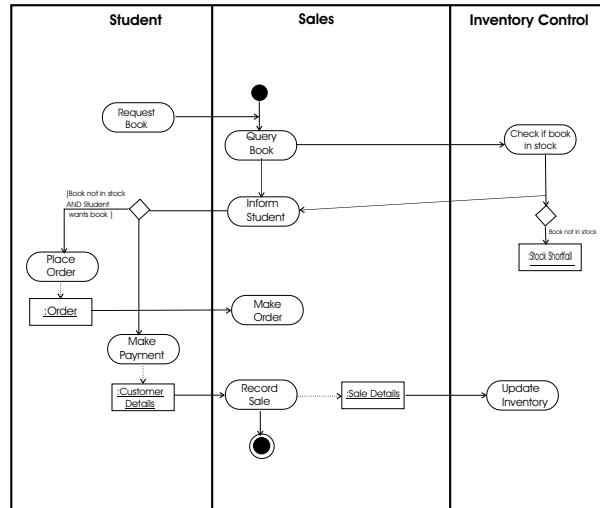
Figs. 5 and 6 give examples of role schemata for the case study.

### 3.4 Environment Model

Roles are embedded within an environment that should be explicitly modelled. The environment consists of passive objects whose states change due either to the activity of other agents or independently of the agents embedded in it. An agent's environment can be completely specified by the change in state of all the objects it can perceive in the environment.

It is important for an analyst to understand and describe the *environment components* (objects and their properties), how frequently objects are generated and how frequently they change (*environmental dynamism*), and which parts of the environment are accessible to which agents (*accessibility*).

The *environment diagram* (see Fig. 7) can be used to show the relationships between roles and the objects in the environment. An arrow from an object (the



**Fig. 3.** Sales Activity Diagram

TASK	Handle a Sale
GOAL	Sell a Book
INPUTS	Required Book Details
KNOWLEDGE RE-QUIREMENTS	Availability and Price of book
OUTPUTS	Payment, Record of Sale, Updated Inventory

**Fig. 4.** Task Schema

rectangles) to a role (a circle) denotes that the role reads the object while the arrows from a role to an object denotes that the role changes or generates the object. Most of the objects in the example domain change relatively infrequently. The diagram also shows the possibility of agents communicating through the environment. This is shown when two agents have access to the same object. The existence of communication paths between roles is also shown.

Given that the environment is composed of passive objects, finer grained modelling of the environment can be done using the techniques offered by object-oriented modelling. For example, activity diagrams and interaction diagrams can be used to model state changes of objects through time. Class diagrams can be used to model the properties of individual objects.

### 3.5 Interaction Model

The interactions between agents is analysed and specified with Agent UML Protocol diagrams [2] [18]. These diagrams show the sequence of messages exchanged between agents.

ROLE SCHEMA:	<b>Seller</b>
GOALS:	Sell a Book
TASKS:	Record Book Request, Query Inventory, Record Customer Details, Obtain Payment, Arrange Delivery, Record Sale
PERMISSIONS:	reads <i>Book Request //Student request for book</i> generates <i>Customer Details</i> generates <i>Sale Record</i>

**Fig. 5.** Role Schema: Seller Role

ROLE SCHEMA:	<b>Inventory Manager</b>
GOALS:	Maintain an Inventory of Stock. Ensure that available books does not fall below threshold values.
TASKS:	Answer Queries. Update Inventory. Arrange for book orders.
PERMISSIONS:	changes <i>Inventory //Database of Books</i> generates <i>Stock Shortfalls</i> reads <i>Sale Record</i> reads <i>ConsignmentNote //Delivered Books</i>

**Fig. 6.** Role Schema: Inventory Manager Role

An inter-agent communication language for specifying these messages needs to be decided upon. A pre-defined one like the FIPA ACL or KQML may be used or a new one can be designed. The choice is likely to depend on whether the agents need to communicate with agents implemented by other organisations.

## 4 Intra-agent Models

The design of individual agents is complicated by the fact that there is no agreement within the agent community on a programming model for agents. The design models are thus necessarily incomplete.

An outside-in approach is used in defining the intra-agent models—the interactions between agents and between agents and their environments is specified first.

### 4.1 Role Instantiation Model

The first step in building intra-agent models is the instantiation of roles. Here the engineer decides on the actual agents that will play certain roles. A role as defined in the inter-agent level may be played by one or several agents. Efficiency considerations may necessitate an agent playing more than one role.

Fig. 8 shows the instantiations of some roles for the case study. The notation is derived from the GAIA Agent Model.

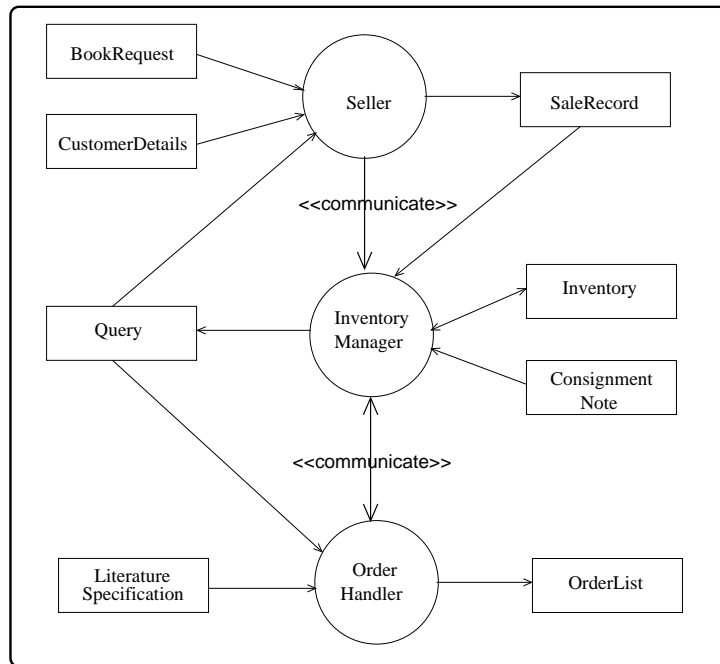


Fig. 7. The Environment

#### 4.2 The Services Model

The engineer can now identify the services that each agent should provide in order to fulfil its task assignments. The protocols and activities necessary to provide these services are then defined.

A service is a “coherent block of activity” [26] that an agent engages in either on behalf of other agents, a human user, or for the MAS. A service is thus characterised by who the agent is willing to do the service for, the activities it is capable of performing, and the protocols it can engage in. A protocol is a pattern of interaction, while activities are actions an agent performs without interacting with other agents.

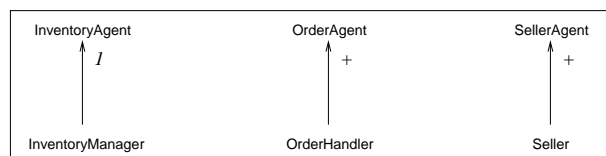


Fig. 8. The Role Instantiation Model

An example of a services model (for the inventory agent) is given in Fig. 9. In the acquaintances column, agents are underlined while protocols are underlined in the Protocols/Activities column. The services model also provides opportuni-

SERVICE:	ACQUAINTANCES:	PROTOCOLS / ACTIVITIES:	INPUT:	OUTPUT:
AnswerQueries	Students <u>SellerAgent</u> <u>OrderAgent</u> Staff	<u>Query</u>	query	answers
UpdateInventory		write inven- tory	Sale Record, Consignment Note	updated inventory
Balance Stock Short- falls		calculate short falls	sale record	short fall list

**Fig. 9.** The Services Model

ties for the engineer to consider the human-computer interface. Here the humans and agents are conceived as cooperating to achieve certain task objectives. The model shows the communication paths between human and artificial agents.

### 4.3 Architecture Definition Models

The designer must provide a detailed design for each agent. In particular, the internal architecture of the agent must be designed at this stage. This process is dependent on the target platform and implementation language.

The following are key issues that need to be addressed:

- **Reasoning requirements.** Each agent’s decision making capabilities need to be analysed. Agents with complex decision making requirements would require a reasoning engine (for e.g. a rule-based system).
- **Reactive vs Deliberative.** The decision on whether to use a reactive or deliberative architecture depends on the reasoning requirements of the agent.

While many architectures have been implemented, they were designed to solve specific problems in particular domains. Their reuse in other domains is questionable. As earlier chapters have shown, there is no agreement on languages for agent-oriented systems. The usual case is that agents are implemented in some other paradigm, with object-oriented technologies being the usual choice.

It is thus envisaged that the detailed designs be done by using established techniques such as those provided by object-oriented methods or functional decomposition.

## 5 Comparison with Other Methodologies

The proposed methodology borrows much from GAIA methodology but differs in significant ways. The use of the inter-agent and intra-agent levels provides a clear

separation of concerns and a structuring mechanism for the models produced. It does not make an early commitment to the nature of agents that eventually populate the MAS, as the specification of agents is left to late in the methodology.

One important difference is that it is task-driven rather than use-case driven. The concept of tasks aids in role identification and it gives more explicit guidance for this key step. Various ways have been used to identify roles. The GAIA methodology gives no procedure or guidelines on role identification and assumes that it follows from the requirements definitions; neither does it give guidelines on requirements gathering. The PASSI methodology first defines use cases; agent and role identification is then a process of grouping use cases. The PASSI domain description phase uses a use-case diagram but each use-case can be seen as task for example *Sell A Book*. The ROADMAP methodology also begins by defining use cases. However, in uncovering use cases, the user is imagined to be interacting with “teams of ideal agents”. It thus makes an early commitment to the types of agents that should exist in the final system.

The other methodologies surveyed (for e.g. MaSE and Tropos) use the concept of goals as a unifying and central concept in the early phases. We suggest that goals are more difficult to identify—it is far easier to know what is being done rather than why. Most of the example goals provided by these methodologies can easily be seen as tasks. For e.g. is ‘sell a book’ a goal or a task?

The task model is similar to the commonKADS task model but its purpose is different. Here, we use it mainly to uncover roles. There are several advantages to using tasks in the early phases. They are easy to identify and describe. They can be uncovered by interviewing stakeholders (who would find it easier to talk about what they do rather than why) and from the organisation’s artifacts (forms, paper trails, etc.).

The use of roles and the organisation metaphor is also not uncommon. The methodology does, however, provide a novel environment model. Few of the methodologies provide explicit models of the environment in which their agents exist. The ROADMAP methodology provides the notion of ‘zones’, but does not show the interaction between agents and objects in the environment. In GAIA and Tropos, environmental information is implicit in other models.

The methodology also differs from other methodologies in that it specifies protocols much later. The agent must be equipped to engage in a protocol and this thus concerns its internal architecture.

The issue of designing for open systems is dealt with in the services model. Here, the engineer describes the interface of the agent to the outside world by specifying what services it provides and to whom it is prepared to provide it. The specification of services rather than tasks allows the engineer to generalise the functionality of an agent.

## References

1. Espen Andersen. Juul Møller Bokhandel a/s. A Case Study available at <http://www.espen.com/papers/jme.pdf>, 2001.

2. B. Bauer, J.P. Müller, and J. Odell. Agent UML: A formalism for specifying multiagent software systems. In P. Ciancarini and M.J. Wooldridge, editors, *Proceedings of the First International Workshop (AOSE-2000)*, volume 1957 of *Lecture Notes in Artificial Intelligence*, pages 91–103. Springer-Verlag, 2001.
3. Birgit Burmeister. Models and methodology for agent-oriented analysis and design. In Klaus Fischer, editor, *Working Notes of the KI'99 Workshop on Agent-Oriented Programming and Distributed Systems*, 1996.
4. Piermarco Burrafato and Massimo Cossentino. Designing a multi-agent solution for a bookstore with the PASSI methodology. In Paolo Giorgini, Yves Lespérance, Gerd Wagner, and Eric Yu, editors, *Proceedings of Agent Oriented Information Systems (AOIS-02)*, Bologna, Italy, July 2002.
5. Jaelson Castro, Manuel Kolp, and John Mylopoulos. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. *Information Systems*, 2002.
6. J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS98)*, pages 128–135, 1998.
7. Stan Franklin. Coordination without communication. web document.
8. G.Caire, F. Leal, P.Chainho, R. Evans, F. Garijo, J. J. Gomez-Sanz, J. Pavon P. Kerney, J. Stark, and P. Massonet. Eurescom p907: MESSAGE:a methodology for engineering systems of software agents. Project report, Eurescom, 2002. available at <http://www.eurescom.de/public/projects/P900-series/p907/>.
9. Norbert Glaser. The CoMoMAS approach: From conceptual models to executable code. In *Proceeding of the 8th European Workshop On Modelling Autonomous Agents in a Multi-Agent World : Multi-Agent System Engineering (MAAMAW-97)*, 1997.
10. Carlos A. Iglesias, Mercedes Garijo, José C. González, and Juan R. Velasco. Analysis and design of multiagent systems using MAS-CommonKADS. In M.P. Singh, A. Rao, and M.J. Wooldridge, editors, *Proceedings of the 4th International Workshop on Agent Theories, Architectures, and Languages*, volume 1365 of *Lecture Notes In Artificial Intelligence*, pages 313–328, Berlin, July 1998. Springer.
11. Nicholas R. Jennings and Jose R. Campos. Towards a social level characterisation of socially responsible agents. *IEE Proceedings on Software Engineering*, 144(1):11–25, 1997.
12. Thomas Juan, Adrian Pearce, and Leon Sterling. ROADMAP: Extending the GAIA methodology for complex open systems. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '02)*, pages 3–10, Bologna, Italy, July 2002. ACM.
13. Thomas Juan, Leon Sterling, Maurizio Martelli, and Viviana Mascardi. Customizing aose methodologies by reusing aose features. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-03)*, Melbourne, Australia, July 2003.
14. Thomas Juan, Leon Sterling, and Michael Winikoff. Assembling agent oriented software engineering methodologies from features. In Paolo Giorgini, Yves Lespérance, Gerd Wagner, and Eric Yu, editors, *Proceedings of Agent Oriented Information Systems (AOIS-02)*, Bologna, Italy, July 2002.
15. David Kinny, Michael Georgeff, and Anand Rao. A methodology and modelling technique for systems of bdi agents. In *Agents Breaking Away — Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-96)*, number 1038 in *Lecture Notes in Artificial Intelligence*. Springer, 1996.

16. Jürgen Lind. *MASSIVE: Software Engineering for Multiagent Systems*. PhD thesis, University of the Saarland, 2000.
17. Allen Newell. The knowledge level. *Artificial Intelligence*, (18):87–127, 1982.
18. J. Odell, V. Parunak, and B. Bauer. Representing agent interaction protocols in UML. In P. Ciancarini and M.J. Wooldridge, editors, *Proceedings of the First International Workshop (AOSE-2000)*, volume 1957 of *Lecture Notes in Artificial Intelligence*, pages 121–140. Springer-Verlag, 2001.
19. L. Padgham and M. Winikoff. Prometheus: A methodology for developing intelligent agents. In *Proceedings of the Third International Workshop on Agent-Oriented Software Engineering*, 2002.
20. Guus Schreiber, Hans Akkermans, Anjo Anjewierden, Robert de Hoog, Nigel Shadbolt, Walter Van de Velde, and Bob Wielinga. *Knowledge Engineering and Management: The CommonKADS Methodology*. The MIT Press: Cambridge, MA, USA, 2000.
21. Daniel P. Siewiorek, C. Gordon Bell, and Allen Newell. *Computer Structures: Principles and Examples*. McGraw-Hill, 1982.
22. Gerd Wagner. Towards agent-oriented information systems. Preliminary report, Institut für Informatik, Freie Universität Berlin, 2000.
23. Gerd Wagner. The agent-object-relational metamodel: Towards a unified view of state and behaviour. *Information Systems*, 2002.
24. Gerhard Weiß. Agent orientation in software engineering. *Knowledge Engineering Review*, January 2002.
25. Mark F. Wood and Scott A. DeLoach. An overview of the multiagent systems engineering methodology. In Paolo Ciancarini and Michael J. Wooldridge, editors, *Agent Oriented Software Engineering. First International Workshop, AOSE 2000, Limerick, Ireland.*, number 1957 in *Lecture Notes in Computer Science*, pages 207–221. Springer-Verlag, Berlin, 2001.
26. Michael Wooldridge, Nicholas R. Jennings, and David Kinny. The GAIA methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, September 2000.
27. Franco Zambonelli, Nicholas R. Jennings, Andrea Omicini, and Michael Wooldridge. Developing multiagent systems: The gaia methodology. *ACM Transactions on Software Engineering and Methodologies (TOSEM)*, 2003.