The Knowledge Engineering Review, Vol. 17:3, 215–260. © 2002, Cambridge University Press
DOI: 10.1017/S0269888902000450 Printed in the United Kingdom

Analysis and design of agent-oriented information systems*

OFER ARAZY and CARSON C. WOO

Faculty of Commerce and Business Administration, University of British Columbia, 2053 Main Mall, Vancouver, BC, Canada V6T 1Z2. E-mail: arazy@commerce.ubc.ca; carson.woo@ubc.ca

Abstract

Analysis and design of Information Systems (ISs) is the process of eliciting the system's requirements and transforming them into a model that can be used to develop ISs. Analysis and design of Agent-Oriented Information Systems (AOISs) relates to the very same process using the multi-agent paradigm. A comprehensive and rigorous methodology for developing multi-agent systems is lacking (Elammari & Lalonde, 1999; Odell *et al*., 2000). Most existing multi-agent systems were developed in an ad-hoc manner, and systems developers paid little attention to requirements specification and the analysis process (Treur, 1999a).

The paper has two goals: (a) to provide an overview and (b) to discuss challenges and future research of the field. To address the first goal, we review different methodologies that are suitable for analysing and designing AOIS. This is done by examining, for each methodology, its suitability in supporting the early phases of the software engineering process (specifically analysis and design) as well as its capabilities for modelling agent-oriented systems. To address the second goal, we analyse the limitations of existing approaches, identify critical issues and point to what we think are possible future directions.

1 Introduction

1.1 Background

Agent-oriented approaches represent an emerging paradigm in software engineering. This emerging paradigm, which is built around a new type of abstraction – an agent – seems to reshape the way new information systems are designed and developed. According to this paradigm, information systems (ISs) are designed as a collection of software agents, which interact with each other in order to realise their goal, thus forming a multi-agent society.

In order to apply the agent paradigm in large and complex industrial applications, and across various domains, tools for creating such systems are needed. One of the basic and most important tools in software development is an Analysis and Design (A&D) methodology. An A&D methodology supports the developer in the early stages of the Software Engineering (SE) process – the analysis and design phases.

Analysis and design methodologies have been widely developed for software engineering. They provide a formal approach to specifying the systems requirements and describing the environment where the system is to be implemented (analysis phase), and to defining the architecture of the future system (design phase). Yet there are no well-established agent-oriented methodologies. There is,

* This research was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada.

however, wide agreement on the importance of developing agent-oriented A&D methodologies (Burmeister, 1996; Elammari & Lalonde, 1999; Kendall *et al.*, 1996; Luck *et al.*, 1997; Treur, 1999a; Wooldridge *et al.*, 1999). This task is one of the top priorities in the effort of turning agent-oriented software engineering into a practical and useful approach, and agent-oriented IS into widely adopted information systems architecture.

The purpose of this paper is, therefore, to review the existing methodologies, models and representations that are necessary or useful in guiding the development of Agent-Oriented Information Systems (AOISs).

1.2 Terminology

1.2.1 Information systems

For the purpose of this work, we consider an information system as a computer-based system for supporting the information processing needs of the organisation. It is not restricted to application systems based on DataBase Management Systems (DBMS) technology. Systems such as those for supporting decision-making (e.g. expert systems and simulation systems) are within its scope, since they also support the information processing needs of the organisation. Throughout this paper we use the terms "organisation" and "business processes" to describe the environment where the information systems will be implemented, and the term "system" to refer to the information system.

1.2.2 Systems analysis and design

An analysis and design methodology supports the processes of eliciting the system requirements, analysing the organisational environment and designing the information system. The early requirements and analysis phases deal with specifying the requirements of the system, while the design phase deals with the detailed specification of how the system will accomplish the requirements. In the analysis phase the application domain is modelled using organisational notions, and the conceptual model is generated.[1] The conceptual model is a model of the organisation and the business processes. In the design phase the information system itself is modelled using technical concepts, which relate directly to components of the software system. The design model should provide detailed information of what the system would look like, but it should not provide instruction as to how to implement the design. The design model is similar to an architect's plan – it describes the exact form of the end product, without specifying the technique and methods that should be used to realise this plan (Mylopoulos, 1999; Wand & Weber, 1990; Wand & Weber, 1993; Wooldridge *et al.*, 1999).

1.2.3 Agent-oriented information systems

There are numerous definitions of agents and multi-agent systems.[2] For the purpose of this work, we will consider the various notions of agents by describing their most important characteristics. Most agent definitions comply with Wooldridge and Jenning's definition of weak agency (Kendall *et al.* 1996, Wooldridge and Jennings 1995b), which describe an agent with the following characteristics:

(a) *autonomous* – agents operate without direct intervention;
(b) *social* – agents interact with other agents, thus forming a multi-agent society;
(c) *reactive* – agents perceive the environment, respond to changes in the environment and affect the environment through their actions; and
(d) *pro-active* – the agent's behaviour is goal-oriented.

[1] The term "conceptual model" has been used in the literature in different contexts. We adopt the definition of Wand and Weber (1989; 1993; 1995), which relates to a conceptual model as a model of the real-world environment (and not a model of the information system). More details on conceptual models are given in Section 2.

[2] The following references deal with the definition of agents and multi-agent systems in detail: Jennings *et al.* (1998), Franklin and Graesser (1996), Klusch (1999a), Bradshaw (1997), Briot and Gasser (1998), Kendall *et al.* (1996a), Wooldridge and Jennings (1995b).

Other characteristics of agents may only appear in some of the multi-agent systems, and thus are not part of the weak agency definition. For example, rationality, veracity, mobility, adaptability and learning capability.[3]

Given the above description of information systems and agents, we consider agent-oriented information systems as multi-agent systems, which consist of autonomous, social, reactive and pro-active agents, for supporting the information processing needs of the organisation. For the purpose of this paper, we also use the terms "agent systems" and "multi-agent systems" to refer to AOISs.

1.2.4 Agent-oriented analysis and design

Analysis and design of Information Systems (IS) is the process of eliciting the system's requirements and transforming them into a model that could be used to develop IS. Analysis and design of AOISs relates to the very same process using the multi-agent paradigm.

From a software engineering perspective, A&D of agent-oriented information systems[4] is not very different from A&D of other information systems. In both, the system analyst and designer has to follow common software engineering practices and elicit requirements, represent the requirements in a conceptual model, design the system's architecture and describe the architecture in the design model.

From a modelling perspective, A&D of agent systems is unique and is based on the agent paradigm, thus both the organisational environment and the information system are represented in terms of agents (human or software) and their interactions. The distinct characteristics of agents and agent systems (see Section 1.2.3 above) require new forms of representation.

We will revisit both these perspectives throughout this paper.

1.3 Motivation for the paper

We believe that a survey of agent-oriented analysis and design techniques is necessary for the following reasons:

1. To date, there is no comprehensive survey that focuses on analysis and design methodologies for agent-oriented information systems. Existing surveys have either studied the general analysis and design methodologies that are not strictly agent-oriented, or they have studied the general agent-oriented frameworks that are not necessarily analysis and/or design methodologies (such as agent-oriented programming languages, implementation architectures or agent-oriented simulation environments).
2. Over the past several years, many frameworks and techniques for developing agent-systems have been proposed. This paper identifies nineteen agent-oriented methodologies and approaches for supporting analysis and/or design. The rapid growth in the field and its growing importance require some organisation in order to understand better what was achieved to date, what still needs to be done and what are the future challenges.
3. It is unclear in existing literature whether a methodology is intended to support the analysis, design and/or implementation processes, and many methodologies mix up organisational and information systems modelling constructs. In addition, given the domain to be modelled in multi-agent systems covers a wide range of aspects (such as the agent's cognitive map, organisational structure and interactions), it is usually unclear which particular aspect a methodology is intended to model. As a result, it is difficult for people with a specific need in mind to select a suitable methodology for their purpose.

[3] Gasser (Briot & Gasser, 1998) gives a slightly different definition of agents, and claims that what differentiates agents from other concepts (such as distributed objects) is the agent's "structured persistent action" – its ability to strive for a goal persistently, while making choices at the time of action.

[4] In this paper we also use the terms "agent-oriented A&D" and "A&D of agent systems"

Our survey is intended to address these concerns and provide an up-to-date analysis of the field from a modelling perspective. We are interested in questions such as: What constructs, relations, and models do existing methodologies employ to represent AOIS (in both the analysis and design phases)? In what areas do existing methodologies struggle to provide appropriate models?[5] Where could we search for solutions to these difficulties?

The paper should allow researchers and practitioners to distinguish between the different methodologies, and understand in what environment each methodology can be best employed. It is clear from the work that there is no best A&D technique and each one is focused on a different aspect of the system.

This paper is different from the surveys done by Yu (1997) and by Iglesias *et al*. (1999) and Tveit (2001). Yu described only methodologies that support the analysis (requirements engineering) phase, and not the design or implementation phases. Iglesias *et al*. did not focus on analysis and design methodologies. Their survey included mainly multi-agent implementation architectures and frameworks that are aimed at assisting the developers at the implementation phase. Tveit's survey is very brief and does not contain in-depth study of the field. It also reviews only a few methodologies. Also, the analysis dimensions that are used in this paper are different and are based on a novel framework. The analysis in this paper is much more comprehensive and detailed. In addition, we include an analysis of the modelling constructs employed for modelling agent systems, and we conclude with a discussion of the major challenges facing this research discipline.

1.4 Scope of the survey

Although A&D methodologies have been used in software systems engineering for a long time and current object-oriented techniques have matured and gained wide acceptance, they are unsuitable for developing agent-oriented information systems. The reason is that agent-oriented systems use different abstractions, and the definition of an object cannot capture the structure and behaviour of agents. The distinctions between objects and agents have been pointed out by many.[6] According to Wooldridge *et al*. (1999), the most important are the agent's flexible, autonomous problem-solving behaviour (the ability of the agent to reason about its environment and choose an action that draws it closer to achieving its goal) and the richness of agents' interactions. An intuitive way of describing the differences is the slogan "objects do it because they have to, while agents do it because they want to (or choose to)".

Due to the aforementioned differences between objects and agents, object-oriented methodologies cannot be used as they are to develop agent-oriented information systems. While some ideas from object-oriented A&D techniques can be borrowed, there is clearly a need for new methodologies, which are rich enough to capture the structure and behaviour of the individual agent (e.g. agent's knowledge, goals, plans and actions) and those of the agent society (e.g. authority and acquaintance hierarchies, and coordination and cooperation mechanisms). Thus we exclude object-oriented techniques from this survey.[7]

In this paper, we review purely agent-oriented analysis and design methodologies. Frameworks that are not strictly analysis and design methodologies (such as agent-oriented development frameworks

[5] By "struggling to provide appropriate models" we refer to certain aspects of AOIS where no sufficient models exist, where there is a lack of agreement in the community on the basic constructs or where there are fundamental modelling problems impeding advancement.

[6] For example, Burmeister (1996), Pont and Moreale (1996), Jennings *et al*. (1998), Wooldridge *et al*. (1999), Wooldridge and Jennings (1995a), Wooldridge and Jennings (1998), Elammari and Lalonde (1999), Shoham (1993), Kendall *et al*. (1996a), Odell *et al*. (2000).

[7] Methodologies and architectures such as F3 (Bubenco, 1993), which are not strictly agent-oriented, are not included, although they could be used to model some aspects of multi-agent systems. Surveys of object-oriented analysis and design methodologies, on the other hand, can be found in OMG (1992), Fowler (1992), Brinkkemper *et al*. (1995), Arnold *et al*. (1991) and Monarchi *et al*. (1992).

and agent-oriented programming languages) are not included in the survey.[8] However, they are discussed in Appendix B, where we review works in relevant neighbouring fields.

1.5 Outline

The paper proceeds as follows. Section 2 lists the works included in the survey and provides some background for each. Section 3 analyses to what extent each methodology supports the analysis and design processes. Section 4 studies the constructs and relations employed by each methodology and the extent to which they cover the features of multi-agent systems. Section 5 identifies major challenges and points to possible future directions. Finally, Section 6 concludes the survey. Appendix A presents a description of each methodology and Appendix B reviews relevant neighbouring fields.

2 Methodologies surveyed

The field of analysis and design of AOISs is still young and it is quite a long way before standard techniques, such as UML in the object-oriented world, will emerge. This research area is fragmented and no one approach is considered a standard. We identified nineteen methodologies, as listed in Table 1.[9] Due to space limitations, we leave the description of each methodology for Appendix A.

Table 1 below gives the list of methodologies included in the survey.

As can be seen from the table, most methodologies are developed in academic institutions, with a few developed by business organisations. Another interesting finding is the dominance of European- and Australian-based techniques.

The methodologies employ as theoretical foundations either Object-Oriented (OO) methodologies (the vast majority) or Knowledge Engineering[10] (KE) frameworks (with one exception). Object-oriented methodologies provide a solid foundation for agent-oriented modelling, since the OO modelling field is mature and these two approaches have many similarities (see Section 1.3). The limitation of OO as a theoretical foundation is that OO methodologies do not provide constructs and models for representing the agent's mental states and for modelling rich social interactions. KE frameworks are more general architectures and are especially suitable for modelling the organisational environment. Their limitation is that they are not tailored for analysis and design of information systems and lack proper support for these processes. These methodologies, similar to OO techniques, are not designed to support agents' social interaction.

In the table above, "High-level/intermediate models", CoMoMAS, and MAS-CommonKADS are based on knowledge engineering foundations, AORBWM borrows from business process modelling, and the remaining fifteen methodologies are based on object-oriented software engineering.

3 The supported software engineering phases

An A&D methodology is used to guide the developing of software systems. Yet, from reviewing works in the field, it is often difficult to identify the software development phases that a particular methodology is supposed to support. Many methodologies use the same constructs and models for

[8] Examples of agent frameworks and implementation tools that are not strictly A&D methodologies are Concurrent METATEM (Fisher, 1996; 1999a; 1999b), AOP (Shoham, 1993) and SIM_AGENT (Sloman, 1999, Sloman & Logan 1999). In Appendix B we discuss the relation between these frameworks and A&D methodologies. A more comprehensive overview of agent-oriented development tools can be found in Reticular Systems (1999) (a survey of commercial products and academic research programmes), and in Wickler (1999).

[9] An additional agent-oriented analysis and design methodology, Tropos (Mylopoulos *et al*., 2000), was not included in this survey. Tropos is a requirement-driven methodology, which is based on the I* architecture (Yu, 1995). A description of Tropos is scheduled to appear in *Information Systems*, Elsevier, in 2002.

[10] Knowledge engineering is the process of domain-knowledge acquisition, knowledge representation, validation and verification, and the construction of a computerised knowledge base. Domain knowledge may refer to an agent's mind (as in the case of expert systems) or to the objective reality.

Table 1 Methodologies included in the survey.

Authors	Institute	Methodology	Reference
Bradshaw *et al.*	Boeing	KaOS	Bradshaw *et al*., 1997
Burmeister	Daimler-Benz	Agent-Oriented Analysis and Design (AOAD)	Burmeister, 1996
Begenti & Poggi	Universita degli Studi di Parma, Italy	B&P	Bergenti & Poggi, 2000
Collinot, Drogoul & Benhamou	University of Paris	Cassiopeia	Collinot *et al.*, 1996
DeLoach & Wood	US Air Force Institute of Technology	MaSE	DeLoach, 1999; Wood & DeLoach, 2000
Elammari & Lalonde	Carleton University, Ottawa, Canada	High-level/intermediate models	Elammari & Lalonde, 1999
Glaser	Loria, France	CoMoMAS	Glaser, 1996
Iglesias, Garijo, Gonzalez & Velasco	Universidad de Valladolid, Spain	MAS-CommonKADS	Iglesias *et al.*, 1996
Kendall, Malkoun & Jiang	Royal Melbourne Institute of Technology	AO methodology for enterprise modelling	Kendall *et al.*, 1996
Kinny, Georgeff & Rao	Australian AI Institute	AAII	Kinny *et al.*, 1996; Georgeff & Rao, 1995
Odell, Parunak & Bauer	James Odell Associates, Siemens and others	AUML	Odell *et al.*, 2000
Moulin & Brassard	Laval University, Quebec, Canada	MASB	Moulin & Brassard, 1996
Pont & Moreale	Leicester University, UK	Integrated Software Engineering (ISE)	Pont & Moreale, 1996
Wagner	Freie Universität, Berlin	AOR (Agent-Object Relationship)	*Wagner 1999; 2000*
Wooldridge, Jennings & Kinny	Queen Mary and Westfield College, London & University of Melbourne	Gaia	Wooldridge, *et al.*, 1999; 2000
Yu, Du Bois, Dubois & Mylopoulos	University of Toronto and University of Namur, Belgium	I* and ALBERT	Yu *et al.*, 1995; Du Bois, 1995
Caire, Leal, Chainho, Evans, Garijo, Gomez, Pavon, Kearney, Stark, & Massonet	EURESCOM (European institute)	MESSAGE/UML	Caire *et al.*, 2001
Gervais & Muscutariu	Laboratoire d'informatique de Paris	ODAC (Open Distributed Application Construction)	Gervais & Muscutariu, 2001
Yu & Schmid	University of St Gallen, Switzerland	Agent-Oriented Role-Based Workflow Modelling (AORBWM)	Yu & Schmid, 1999

representing both the business environment and the information system, and do not make a clear distinction between analysis and design.[11] An important distinction between the various methodologies is the Software Engineering (SE) phase that they support (Mylopoulos, 1999; Treur, 1999a). In this section we will map each methodology accordingly and indicate whether or not it differentiates clearly between the phases.

Although various A&D methodologies use various names to describe the models and the phases, there is generally agreement on the structure of the A&D process. It is seen as a collection of transformations, from the very abstract model (the conceptual model) to the most concrete model (the detailed design model), where each transformation shrinks the space of possible end products and introduces more and more implementation bias. Ideally, the transformations should be straightforward, i.e. leave no degrees of freedom to the designer, but this is not the case in most methodologies (Wand & Weber, 1990; Wand & Weber, 1993; Wooldridge *et al.*, 1999).

At the heart of the analysis and design process, there have to be at least two models – the conceptual model and the design model. A&D methodologies include these two models, as well as descriptive and procedural information on how to use the models. A model includes a set of constructs, and relationships between the constructs. The conceptual model includes constructs and relations that capture the static and dynamic nature of the organisation and its environment. It should describe the various entities inside the organisation and those external to it, the different structures and hierarchies in the organisations as well as the business process. The design model, on the other hand, includes constructs and relations that describe the structure and behaviour of the information system. The models should describe the multi-agent system as a whole, as well as the internal components of the individual software agent.

We define each of the SE phases below, and then map the surveyed methodologies according to the phases they support.

- **Analysis** (also referred to as "requirements engineering", "strategic modelling" (Klusch, 1999b, p. 14) or "enterprise modelling" (Fox *et al.*, 1998)) This phase specifies the boundaries of the domain and the business goals and policies of the system, and sketches the organisational environment. The end result is a conceptual model (constructed using abstract entities and not concrete data entities) that describes the organisational processes and organisational entities, and their relations. The conceptual model should answer questions such as, "How will the new system help us realise the business goals?", "How will the new system fit in with the work processes?", "What organisational units and external entities will interact with the system?" and "How will the work processes be realised after implementing the system?"
- **Design** the design phase (also referred to as "formal specifications" phase) specifies the architecture of the future information system that will be built according to the definitions in the conceptual model. The end result of this phase is a design model, which is "an accurate and formal statement that expresses the desired behaviour of the information system" (Coltell & Chalmeta, 1999, p. 2). The design model should be implementation-independent, and it should answer questions such as "What data types will be used?", "What will be the information flows within the system?" and "What will be the attributes that define the internal structure of the agent?"
- **Implementation** This phase specifies the programming languages, hardware platforms, transfer protocols and communications protocols.

In Table 2 the cells marked in grey indicate the SE phases that a particular methodology supports.

Table 2 below also illustrates whether a methodology clearly distinguishes between the analysis and design phases and whether it provides distinct constructs for modelling the organisational and the

[11] Although in many cases constructs such as "agent" or "object" could be employed in both analysis and design, these constructs represent different entities at the different SE phases – in analysis they represent real-life entities, while in design they represent elements of the information system. It is important that methodologies make this distinction clear and provide distinct models for modelling the organisation and the information system.

Table 2 The supported SE phases for A&D methodologies.

Methodology	**The supported SE phases**		
	Analysis	Design	Implementation
KaOS		■	
AOAD	■	■	
B&P		■	
Cassiopeia		■	
MaSE		■	
High-level/intermediate models	■	■	
CoMoMAS	■	■	■
MAS-CommonKADS	■	■	
AO methodology for enterprise modelling	■	■	
AAII		■	
AUML	■	■	
MASB	■	■	
ISE	■	■	
AOR	■		
Gaia	■	■	
I* and ALBERT	■	■	
MESSAGE/UML	■	■	
ODAC	■	■	■
AORBWM	■	■	

information system. We use a thick separating line (▮) to indicate that the phases are clearly distinguished.

As shown in the Table, only 6 out of the 19 methodologies make clear distinctions between the different SE phases. One possible explanation of this is that many methodologies were originally developed to design systems assuming the organisational context and requirements were known. They were later extended to model systems requirements without considering the different representations needed in the two phases. This might also explain why 5 of the design methodologies do not provide constructs and models to capture the organisational environment (in the analysis phase).

Although the focus of this paper is on analysis and design, and we have excluded methodologies aimed at supporting implementation, it is interesting to note that two methodologies (CoMoMAS and ODAC) further extended the framework to support implementation. We believe that this is an indication of the maturity of the field.

It is also interesting to point out that one methodology (AOR) is strictly for supporting analysis. We believe that a strict analysis framework has limited utility for developing information systems and eventually needs to be extended to support systems design.

In the following section, we will further analyse the methodologies according to how many multi-agent systems features they cover. We will then show how the analysis can be combined with Table 2 in this section to highlight the uniqueness of each methodology.

4 Coverage

The second dimension for mapping the methodologies is the degree to which the methodology provides constructs and models for covering the wide spectrum of multi-agent systems features. Since no one has established an agreed-upon set of constructs for this purpose and no applicable ontology has been identified, we offer a framework for studying representational issues of agent systems. We divide the multi-agent "representational space" into four sub-domains, and then for each of these sub-domains we identify a set of construct clusters that are used (e.g. planning, belief and knowledge, and communication). We take a bottom-up approach to study the representational components of existing agent-oriented A&D methodologies. We then map existing methodologies according to their coverage of the construct clusters, in each of the representation sub-domains. Lastly, we discuss the findings from our coverage analysis and try to provide some insights.

4.1 Multi-agent taxonomy

To study the representation models employed by the methodologies, we first need to agree on a set of appropriate constructs. This is a critical issue, since no such set of constructs has yet been identified. Jan Treur states (1999a, p. 41):

> It is crucial that the basic principles and lessons of software and knowledge engineering are applied to the development and deployment of multi-agent systems. At present, the majority of existing agent applications were developed in an ad hoc fashion – following little or no rigorous design methodology and with limited specification of the requirements or design of the agents or of a multi-agent system as a whole.

There is wide agreement on the need for a methodology that will overcome these limitations (Luck *et al.*, 1997). The field is relatively young and it would take quite some time before an acceptable standard, such as UML in the object-oriented world, would emerge (Treur, 1999a; 1999b).

One of the organisations that have been working towards realising this goal is AgentLink, the European Network of Excellence for Agent-Based Computing. AgentLink has a special interest group (SIG) on Methodologies and Software Engineering for Agent Systems. This group, headed by Franco Zambonelli,[12] is trying to draw a road map leading to standards in agent-oriented analysis and design. In a series of meetings (Treur, 1999a; Treur, 1999b; Zambonelli, 2001; Luck *et al.*, 2001; AgentLink, 2001) they have defined short-term, medium-term and long-term objectives for developing standards in multi-agent software engineering. This paper is complementary to the work of AgentLink and could serve as a basis for the next step in the SIG's mission.

If no standard exists, then how can we come up with a set of constructs and relations necessary for representing the complexities of agent systems? One possible approach is a top-down analysis, which is based on an ontology[13] – a formalised conceptualisation of a domain. Wand and Weber (1990; 1993) explored, in a series of works, the use of Bunge's ontology for guiding the development of object-oriented A&D methodologies. To date no such attempt has been made in the agent world, and no ontology has been identified as a suitable candidate for modelling agent systems. Clearly we need an alternative approach for the purpose of this paper.

Below we propose an alternative approach for studying the representation of multi-agent systems. The proposed framework is based on two complementing types of analysis. First we conduct a top-down study by reviewing the literature and identifying several important agent-systems sub-domains that need to be represented. Next we apply a bottom-up approach and study the existing agent-oriented A&D methodologies.

Two ways of categorising multi-agent systems are common in the literature: (a) the individual agent (the internal, micro view) versus the system/society of agents (the external, macro view) (Brazier *et al.*, 1997b; Ciancarini *et al.*, 1999; Elammari & Lalonde, 1999; Sloman & Logan, 1999; Wooldridge *et al.*,

[12] Zambonelli took over from Jan Treur.
[13] Ontology is the study of the things that exist in reality, and has traditionally been studied within philosophy.

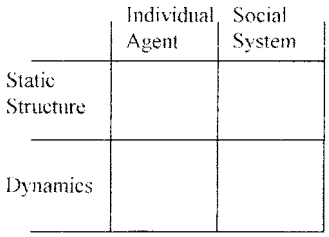

Figure 1 Taxonomy for multi-agent systems

1999; Yu *et al*., 1995), and (b) the dynamic aspects (behaviour) versus the static aspects (structure) (Carley & Gasser, 1999; Research Group AI, Vrije University; Wand & Weber, 1990; Wand & Weber, 1995). The integration of both categorisations forms a two-by-two matrix, where each of the four cells represents one aspect of the multi-agent system, or what we term "a representation sub-domain" (see Figure 1). Although we have not encountered such a taxonomy before, we believe that this is useful in thinking about agent systems and can also be used for people in the field to situate their work.

The top left cell refers to the *Individual agent/static structure* sub-domain. Models, constructs and relations in this sub-domain describe how the internal (cognitive and emotional) map of the agent is structured. The bottom left cell refers to the *Individual agent/dynamics* sub-domain. Models and constructs in this sub-domain deal with the functionality of the agents: the evolution of the agent's knowledge and goals, planning and learning. The top right cell refers to the *Social system/static structure* sub-domain. Models and constructs in this sub-domain describe the structure within the agent society (e.g. organisational structure, role hierarchy and goal hierarchy and the task-resource, task-skills, and roles-tasks relations), and the normative component of the social system, such as permissions, rights and norms. The bottom right cell refers to the *Social system/dynamics* sub-domain. Models in this sub-domain represent the interactions and communications between agents, and the way in which agents coordinate their activities, cooperate and compete. Together these four sub-domain compose the "representational space" of agent systems,[14] which we will use to analyse the coverage of each methodology later.

4.2 Concept clusters

Below we describe the concept clusters we have identified in each sub-domain. These clusters were constructed by first analysing the constructs and relations employed by the methodologies surveyed in this paper, mapping them onto the sub-domains and then identifying a set of concept classes or clusters in each. In addition, we use the KaOS methodology to illustrate the coverage in each sub-domain.

Individual agent/static structure

The concept classes listed below are used to describe the internal structure of the agents.

- **Perception** – what events in the world the agent is capable of perceiving.
- **Knowledge** – the agent's knowledge and **beliefs** about the world, himself and other agents.
- **Transparency** – what parts of the agent's knowledge he is willing to **expose** to other agents.
- **Emotions** – the agent's emotional state (rarely used).
- **Goals** – a list of agent purposes, goals, **preferences** or **desired** states (possible with value or utility associated with each).

[14] Additional important aspects of analysis and design that could be thought of as domains, yet are not included in the analysis, are *Time* (Schobbens & Petit, 1999) and *Uncertainty*. These two concepts relate to aspects that are present in every real-life system and every environment, and many times these concepts are represented in models of analysis and design methodologies. Although these concepts are important, they are not an inherent part of multi-agent systems and the degree to which they are needed depends greatly on the application domain. Thus we did not perform an analysis of the methodologies according to the Time And Uncertainty dimensions.

- **Resources** – a set of resources, skills or capabilities the agent has access to.
- **Actions** and **communications** – a set of actions the agent is able to perform. Communications are a specific type of action, which involves information exchange with other agents.
- **Plans** – a set of possible plans for achieving the goals. Each plan includes a pre-defined set of actions and the required resources.
- **Intentions** and **commitments** – triggered plans (also referred to as commitments to perform an action).

We will use the KaOS methodology (Bradshaw *et al.*, 1997) to explain this representational sub-domain. KaOS includes the following attributes of the agent's cognitive model: knowledge (beliefs and facts), desires, intentions, and capabilities. Facts represent beliefs in which the agent has confidence; facts and beliefs may be held privately or be shared. Desires represent goals and preferences that motivate the agent to act, and intentions represent a commitment to perform an action. However, transparency and perception constraints, and constructs that relate to the emotional state of the agent, are not included in this model. The coverage of KaOS for this sub-domain is thus medium.

Individual agent/dynamic
The concept classes listed below are employed to represent the events that trigger changes in the agent's internal structure.

- **Acting** – events that trigger an agent to (re)act.
- **Reasoning** (for rational agents) – a mechanism for choosing the optimal plan for obtaining a goal based on the possible set of actions, the costs associated with each, and the utility associated with the goals.
- Activating a plan – the events that trigger a plan (and turn it into an intent for actions).
- Activating an intention – the events that trigger an intention, causing the agent to (pro-)act.
- **Learning** and **evolution** – the events that trigger changes to the agent's knowledge, goals or plans.

In KaOS, the agent's dynamics model includes a description of how the cognitive map is updated as the agent goes through its life cycle: birth, life and death. During their lives agents receive, process and send information continuously and according to the information they receive they may update their internal structure. However, there is no description of a planning or scheduling mechanism, and the agents seem to be acting directly on the basis of their intentions, without considering the implications of their actions and other agents' actions on the environment. Thus KaOS's coverage here is medium.

Social system/static structure
The system's structure could be represented with the following set of construct classes.

- **Resources** – the existing resources, skills and capabilities in the system and their organisation (i.e. resource hierarchy).
- **Goals** hierarchy – goals and their (de)composition. May also include a description of the pre and post conditions for the goals.
- **Actions/Tasks/Services** – the possible actions in the system and their organisation (i.e. task decomposition or activities hierarchy).
- Agent **classes** – describing the different classes of agent in the system and (hierarchical) relations between classes.
- **Roles** – a hierarchy of agent's roles and the agent classes associated with each role.
- **Permissions** – the **rights** to use resources (for agent classes or roles).
- **Responsibilities** – the functionality (i.e. actions) and goals associated with roles or agent classes.
- **Acquaintance** (or **friendship**) network – a network of relations between agent classes or roles that define the group of agents an agent will be interacting with (i.e. communicating and sharing information).
- **Authority** hierarchy – the authority and **control** structure in the system (between agent classes or roles), which define for each agent his subordinates and superordinates.

- Global **constraints** – transparency and perception constraints (see "Individual agent/static structure") on agent classes or roles.
- **Dependency** relations – system-level constraints describing dependencies between agent classes or roles (dependencies for using resources, performing tasks or achieving a goal).

Using the same example given earlier, the definition of the social structure of KaOS is very loose. Five generic types of agent are defined (KaOS agent, Mediation Agent, Proxy Agent, Domain Manager and Matchmaker) and aside from the class hierarchy, no model of how the agents relate to each other (such as friendship networks and authority networks) exists. Another form of the system's structure, which is missing in this model, is the hierarchies of tasks, skills and resources, and the description of how all these relate to one another, to agents and to the roles agents play. KaOS coverage of this sub-domain is, therefore, low.

Social system/dynamics

The system's behaviour is emerging through the individual agents' behaviour, and system-level constraints (on the behaviour) are described under "Social system/static structure". Thus under "Social system/dynamics" we include the description of the possible interactions between agents, and agents' cooperation.

- **Messages** – possible messages, including the sender and the receiver, and the contents of the message (many times defined in terms of speech acts).
- **Conversations** – conversations and **communication protocols** define the sequence of messages that make up a conversation.
- **Coordination –** coordination and **conflict resolution** mechanisms that are necessary to enable agents' **cooperation** (could be based on responsibilities, acquaintance and authority of agents).
- **Mobility** (for mobile agents) – agents' **location** over time, and the events that trigger location change.

Social interactions are the main focus of the KaOS architecture, and it provides a rich and comprehensive mechanism for defining the ways in which agents interact. The interaction model includes the speech (illocutionary) acts and the sequencing of the messages. The speech acts are similar to those of KQML (Finin *et al*., 1997), only they are more general, and hence KaOS could be considered as a communication meta-architecture. Speech acts of any agent communication language could be implemented within KaOS. Message sequences are organised in conversations – a pattern of messages transferred back and forth between (two) agents – which are modelled using state transition diagrams. Sequences are pre-defined, and hence when designing a KaOS architecture, the designer should elicit all the possible interaction sequences. Thus KaOS's coverage for the "Social system/ dynamics" sub-domain is high.

4.3 Mapping methodologies by coverage

Above we described a framework for studying the constructs and relations used for agent systems. Below we proceed to map the methodologies according to that framework. We evaluate each methodology on the extent to which it covers the concept clusters described above. We define three categories of coverage:

- High – the methodology provides a comprehensive set of constructs, and the models include complex hierarchies and relations. For example, complex models of agent societies will include hierarchies of roles, authorities, responsibilities and interactions between those hierarchies. This enables a structured modelling process, yet is rather difficult and time-consuming.
- Medium – the methodology provides a pre-defined set of constructs and relations, but the number of supported constructs is rather limited. Some methodologies allow adding constructs on top of the restrictive pre-defined set. Medium coverage methodologies are usually designed for a specific application domain, and allow for a relatively easy and straightforward analysis and design process.

The best examples of such a model are methodologies based on BDI[15] architecture (Georgeff & Rao, 1995; Kinny *et al*., 1996). The set of constructs available for modelling the agent's internal structure is very limited, which enables the agent to formulate a tractable plan.[16]

- Low (but expandable) – the methodology does *not* provide a pre-defined set of constructs, but supports the adding of constructs and relations. These methodologies are very general and can be applied to different application domains. An example is Gaia's (Wooldridge *et al*., 2000) model of the agent's internal structure – in order not to commit to specific constructs and leave much flexibility to the designer, Gaia does not pre-specify constructs. This type of methodology enables an undirected and flexible analysis and design process, which is more appropriate for experienced designers.

The following legend is used in Table 3:

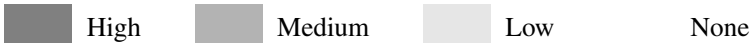

Table 3 Coverage of construct clusters in the representational sub-domains.

Methodology	**Individual level**		**System level**	
	Structure	Dynamics	Structure	Dynamics
KaOS				
AOAD				
B&P				
Cassiopeia				
MaSE				
High-level/intermediate models				
CoMoMAS				
MAS-CommonKADS				
AO method. for enterprise model				
AAII				
AUML				
MASB				
ISE				
AOR				
Gaia				
I* and ALBERT				
MESSAGE/UML				
ODAC				
AORBWM				

[15] In the BDI model agents have beliefs, desires and intentions. These three constructs are used to define how agents perceive the world and the mechanism they employ to choose actions based on their knowledge of the world.

[16] Tractable models are models that can be solved by computers in a finite time and using finite storage spaces.

In Table 3 we map each methodology according to its coverage, as explained above, in each of the representation sub-domains. On top of the three categories defined earlier – high (a pre-defined and comprehensive set of constructs and relations), medium (a pre-defined, but limited, set of constructs and relations), and low but expandable (no pre-defined constructs, ability to define constructs) – we add one more category: *none*. We use this additional category when the methodology does not provide any constructs or models for representing a specific sub-category. Further information about the constructs, relations and models the methodologies employ for mapping each sub-domain is included in Appendix A.

Table 3 provides a summarised view of all 19 methodologies on one page. This could be of great assistance to practitioners in choosing a methodology for a specific application domain. It also provides insights into the state of the field. We will elaborate both of them below.

This comparison dimension seems to be effective – only a few methodologies present a similar pattern. In specific cases where methodologies provide similar patterns, we can combine the first analysis dimension (the supported SE phase) to make the differences clear. For example, AUML and B&P present similar patterns in the table above, but when consulting Table 2 we notice that AUML can be employed to support both the analysis and design phases, while B&P supports the only the design phase.

How can these maps be used to help the system developer in choosing an appropriate methodology? We illustrate this with two examples. Assume that a researcher is studying human social interaction and is looking to develop an agent-based simulation. The researcher is especially interested in tools to support the analysis and modelling of the interaction of a group of people. The methodology best suited for the researcher's needs would support the analysis phases (first analysis dimension) and provide high coverage models for representing societal-level dynamics. "High-level/intermediate models" seems to be a good choice for that specific need.

A second example would be a company looking to automate their manufacturing processes by utilising agent technology. The company is looking to develop a scheduling program based on market-driven agent interaction. Each part and each machine on the manufacturing floor will be represented by agents. An appropriate methodology for this problem would support the design phase (most critical), to a lesser degree the analysis phase, and possibly the implementation phase. The software agents in this system will need to formulate a plan and collaborate with each other. Thus it is essential that the supporting analysis and design methodology employed will provide tractable models for describing the dynamics of the individual agents (specifically planning) and of the agent society (specifically agent communication and conflict resolution mechanisms). "I* and ALBERT" seems to be a good choice of methodology for these purposes.

On a more general level, we obtain the following insights from the table:

- There is agreement that agent-oriented A&D methodologies should put emphasis on system/societal aspects, and many of them cover these aspects well. This could be due to the availability of some commonly accepted techniques or approaches such as communication protocols.
- There are two different viewpoints covering the individual agent aspect. Some methodologies leave the individual aspects open-ended while others provide a very comprehensive set of constructs to model them. This indicates a lack of convergence on the necessary constructs for modelling an agent's internal structure and behaviour. We will revisit this point again in Chapter 6 when we discuss critical issues and future challenges in the field.
- It is interesting to note that approaches based on knowledge engineering as a theoretical foundation (High-level/intermediate models, CoMoMAS and MAS-CommonKADS) seem to include the most extensive set of constructs and relations and the most comprehensive models. We conjecture that it is difficult to extend the object-oriented concepts to the modelling of agents, and that the knowledge engineering field provides a good foundation on which to overcome this challenge.

Up to now we have described the process of agent-oriented systems analysis and design and reviewed existing methodologies. We have highlighted some of the differences between the existing approaches and tried to provide guidelines on when (i.e. in what application domain) each methodology would be

more appropriate. This part is relevant to both practitioners and academics. In the following chapter, we will discuss critical issues and major challenges to this field of research. The discussion is aimed at the group of researchers working on the development of analysis and/or design methodologies for agent-oriented systems.

5 Critical issues and possible future directions

Below we discuss critical issues for analysing or designing agent systems. We will discuss general issues of software engineering and issues related to modelling agent systems.

Below we describe three fundamental issues of software engineering. Although these issues are not unique to analysis and design of AOIS and we have already discussed them in Section 3, we feel it is important to highlight them since many existing methodologies struggle with these issues.

- *Covering and distinguishing the analysis and design phases*. Many existing methodologies struggle with supporting both the analysis and design phases, and with making the distinction between the phases clear. We believe that it is essential to provide tools for capturing the complexity of the organisational environment (conceptual models) and for designing heterogeneous and complex multi-agent systems (design models), and that these phases should be clearly distinguished by providing distinct representation models for each.
- *Guiding the analysis and design process*. Many methodologies do not provide explanations and guidelines on using the diagrams and models and on how to proceed from the conceptual model to the design model (Wand & Woo, 1999; Yu *et al.*, 1995). We believe that it is important to guide the systems analyst and designer through the A&D process (e.g. provide clear guidelines for deciding what entities should or should not be represented by agents and what types of agent they should be, and for relating agents in the analysis models to agents in the design).
- *Approaching design from requirements*. The software engineering process is not linear, and there is much interaction between the SE phases: analysis, design and implementation. Yet it is important that the process be driven by the requirements. This is especially crucial in large-scale systems development when different teams work on different aspects or phases of the development. Several methodologies either do not support the analysis phase or allow description of system specifications without describing first the requirements.

We believe that the issues described above are critical and it is essential for agent-oriented A&D methodologies to address them.

From a modelling perspective, the major obstacle to progress in the field is the lack of convergence on the constructs needed to represent agent systems. We discussed this problem earlier, and the difficulty of defining such a set of constructs. In Section 4.2 we tried to make a contribution in that direction, by conducting a bottom-up analysis and identifying the constructs employed by existing agent-oriented methodologies.

Below we list some areas where the lack of agreement on a set of constructs is especially acute. We also point to some possible future directions, based on insights we have gained from reviewing neighbouring fields.[17]

- *Modelling the agent's internal (or cognitive) structure*. There is no theory to guide the selection of constructs and models for representing the agent's knowledge, goals, plans and learning mechanisms. Existing approaches are guided either by implementation issues (for instance BDI architectures, which are designed for tractability) or by common sense.

[17] These neighbouring fields are agent-oriented programming languages and control architectures, agent-oriented simulation and automatic prototype-generation frameworks, DAI agent-oriented problem-solving frameworks, and Computational Organisational Theory (COT). In Appendix B we describe these research fields and explain how ideas from these areas might be relevant to the modelling of agent systems.

Possible theoretical foundations could be cognitive psychology and cognitive anthropology, specifically the design of cognitive architectures.[18] Bordini (1999) claims that the cognitive approach to anthropology is argued to be a suitable theoretical foundation for the development of multi-agent systems. Fieldwork practice in social anthropology is also indicated as a useful source of ideas. This is specifically relevant to the design of the model of the agent's cognitive (and emotional) structure. Many methodologies borrow from psychology and use in their psychological constructs such as "beliefs", "intentions" and "attitudes", but adoption of a more comprehensive psychological framework is lacking. Ad hoc approaches that are based on common sense provide a good starting point, but ultimately we should build on solid theories of human behaviour. These theories provide a comprehensive and detailed model of human cognition, emotion, and action. The work of Carley and Gasser (1999) tries to link the agent's cognitive architecture to the agent's capabilities and the tasks it needs to perform. This work is based on an earlier work by Carley and Newell (1994), and is an excellent source for understanding the complexity associated with modelling the agent's cognitive architecture.

An example of a framework from a neighbouring field (agent-oriented simulation and automatic prototype generation frameworks; see Appendix B) that builds extensively on knowledge from psychology and formal social theories is STEAM (Tambe, 1997).

- *Modelling agent society's structure*. Agents represent a new abstraction, more complex than objects, and in many ways software agents resemble human agents. To capture the structure of this complex system, constructs – such as roles, authorities and rights – are used. Agent-oriented A&D process should be thought of as a process of constructing a society of agents (Carley & Gasser, 1999; Ciancarini *et al*., 1999; Wooldridge *et al*., 1999). The social approach should be employed at both the analysis phase (view organisational entities and processes as interactions between autonomous agents) and the design phase (design software entities that behave similarly to human agents).

  To date, agent-oriented methodologies employ social concepts (for example acquaintance networks and authority hierarchy); however, none of these methodologies builds on solid sociological theories. Current use of social constructs is in an ad hoc manner, with no reference to models of human social structure. We believe that we should draw on formal social theories and employ knowledge from anthropology and organisational sciences when designing an agent-oriented A&D methodology.

  An example of a framework from a neighbouring field (computational organisational theory; see Appendix B) that provides comprehensive models for describing the agent's society structure is TOVE (Fox *et al*., 1998).
- *Modelling the agent society's behaviour.* If agent societies are represented in terms of human societies, then constructs and models for representing the system-level interactions should be based on models of human social behaviour. If agents are to compete, cooperate, share resources and coordinate their activities, then abstraction for representing these notions is required.

  As with the modelling of an agent's system structure, existing methodologies define constructs in an ad-hoc manner. We believe that theoretical foundations are necessary for modelling the behaviour of agent societies, and again we point to the fields of organisational science and anthropology as possible candidates.
- *Developing precise relations between the organisational structures and local specifications*. Models of the system should be related to models of the individual agent's structure and behaviour. Hierarchies at the organisational level could be used to restrict agent behaviour, for example the friendship network could specify which other agents the agent cooperates with.
- *Enabling agent interactions in open environment*. As open environments become ever more popular, methodologies should support development for borderless heterogeneous systems. To allow agents to interact effectively and cooperate in open and heterogeneous environments two challenges have

[18] University of Michigan (1999) includes a comprehensive study of the theory in this research field, as well as descriptions of specific architectures.

to be addressed: (a) enabling semantic interoperability, so agents designed by different groups can "understand" each other, and (b) enabling interoperability through coordination languages and conflict resolution mechanisms, so agents with conflicting goals can arrive at an action. Both of these challenges are difficult and no complete solutions exist. To solve the first problem, translation mechanisms are used. While translating between languages is a relatively simple task, translation between ontologies poses a much greater challenge (for more details see Genesereth (1997)). Existing solutions for the second challenge include conflict resolution mechanisms (for instance Barber *et al*. (1999)).

Other discussions of challenges in developing agent-oriented software engineering methodologies can be found in Brazier *et al*. (1999) and in Yu (1997).

6 Concluding remarks

As agent technology is gaining greater acceptance and multi-agent systems are becoming increasingly prevalent, there is a growing need for tools and methodologies to support the development of agent-oriented information systems.

In this paper, we have reviewed the existing techniques for supporting the development of agent-oriented systems. The field is currently in its early stages of development. There is no existing methodology that can support both analysis and design, and effectively cover all constructs necessary for modelling agent systems.

Although there are some fundamental challenges facing this research community and we are yet far from reaching an accepted standard for modelling agent systems, several approaches are gaining the leading position, based on the functionality of the methodologies and the reputation of the research team. We identify four such approaches:

- Methodologies based on knowledge engineering, and specifically on CommonKADS (part of the European Community ESPIRIT programme). The two examples we mentioned in this paper are CoMoMAS (Glaser, 1996) and MAS-CommonKADS (Iglesias *et al*., 1996). The strengths of these approaches are in modelling the dynamic aspects of AOIS.
- Methodologies based on the BDI architecture. Example are AAII (Georgeff & Rao, 1995; Kinny *et al*., 1996) and "agent oriented methodology for enterprise modelling" (Kendall *et al*., 1996). The power of these methodologies is in modelling agent's planning capabilities and in providing a tractable mechanism for computing plans.
- Extensions to UML. The most notable examples are AUML (Odell *et al*., 2000), MESSAGE/UML (Caire *et al*., 2001), B&P (Bergenti & Poggi, 2000), and MaSE (DeLoach, 1999; Wood & DeLoach, 2000). These methodologies build on existing object-oriented standards and extend them mainly to modelling agents' interactions.
- Gaia, a methodology by Wooldridge, Jennings and Kinny (1999; 2000), which makes the analysis and design process clear and structured, and provides rich models to capture the structure of agent societies.

This work makes several contributions. First, this is by far the most comprehensive survey of the field. The paper provides numerous references to works in the field of agent-oriented analysis and design, and in related fields. Thus it could serve as a valuable resource for both practitioners and academics.

Second, we define the analysis, design and implementation phases, and present a mapping of agent-oriented analysis and design methodologies accordingly. In doing so, we identify methodologies that follow good software engineering practices, making a clear distinction between the models aimed at capturing system's requirements and models for representing the information system.

Third, we develop a framework for studying the modelling of agent systems. We decompose the agents "representational space" into four sub-domains and, through a bottom-up analysis, identify important clusters of constructs for each sub-domain. In addition, we provide a mapping of each methodology according to its coverage of the construct clusters. The framework we develop could be

used to situate and differentiate existing works. Furthermore, researchers in the field can use the set of identified construct clusters to guide discussions on the convergence of a set of constructs.

Finally, we identify critical issues facing researchers in this field and point to possible solutions.

Appendix A: description of the methodologies

Appendix A includes a description of each of the methodologies described in Section 2. Each description includes an overview of the methodology, its aim and the theoretical and logical foundation it employs.

We employ the framework described in Section 4 for decomposing the representation domain of multi-agent systems into four sub-categories. We evaluate the modelling capacity of each methodology in each of the four representational sub-categories. The summarised result of this evaluation process is presented in Table 3 in the main body of this document.

Following is a description and an evaluation of the agent-oriented frameworks. Since experimenting with all the methodologies would require a lifetime of effort, we evaluated the methodologies based solely on the description provided in the referenced sources. It is possible that certain features that appear to be missing actually exist in the methodology and were simply left out of the source. It is also possible that in the most recent version of the source, the methodologies have been updated or enhanced.

KaOS; Bradshaw *et al.*

KaOS is an architecture for designing agent systems that focuses on the communications among agents, and hence can be considered as an open agent communication meta-architecture (Bradshaw *et al*., 1997). It aims at providing an infrastructure for implementing and integrating diverse types of agent-oriented systems. The KaOS architecture is neutral with respect to hardware platforms, operating systems, transport protocols, programming languages and the type of communication primitives used.

The design of the system is based on object-oriented programming approaches where agents are treated as "objects with intentions". A hierarchy of agent classes exists, and each agent is an instantiation of a specific class. Agents inherit attributes based on the class hierarchy.

Individual agent/static structure

KaOS includes a relatively simple model of the agent's cognitive map, which includes the following attributes: knowledge (beliefs and facts), desires, intentions and capabilities. Facts represent beliefs in which the agent has confidence about; facts and beliefs may be held privately or shared. Desires represent goals and preferences that motivate the agent to act, and intentions represent a commitment to perform an action. There is no exact description of how these attributes are related to each other or of how the cognitive map leads to the agent's actions.

Individual agent/dynamics

The agent's dynamics model include a description of how the cognitive map is updated as the agent goes through its life cycle: birth, life and death (there is also a cryogenic state) (Bradshaw *et al*., 1997). During their lives agents read, process and send information continuously, and according to the information they receive they may update their internal structure.

There is no description of a planning or scheduling mechanism, and the agents seem to be acting directly on the basis of their intentions, without considering the implications of their actions and other agents' actions in the environment.

Social system/static structure

The definition of the social structure within KaOS is very loose. Five generic types of agent are defined (KaOS agent, mediation agent, proxy agent, domain manager, and matchmaker) and, aside from the class hierarchy, no model of how the agents relate to each other exists (such as friendship networks, authority networks, etc.).

Following is a short description of each one of the agent types used in the model.

- *KaOS agent* is defined according to the attributes specified above, and is the main component of any agent system. This is the agent that actually performs the work the system is designed to do.
- *Mediation agents* provide interface between KaOS agents and external entities.
- *Proxy agent* is a special case of mediation agent, which provides interface between two KaOS agent domains.
- *Domain manager* controls the exits and entries of agents into the domain (according to pre-defined guidelines), registers all the agents in the domain and keeps a list of their current addresses.
- *Matchmaker* is a special case of mediation agent, and is responsible for registering the services each one of the agents provides. This agent receives requests for service from the KaOS agents and matches them with the agent that provides the requested service.

In this architecture only a very simple type of broker is used as a middle agent within the KaOS agent domain – the matchmaker. Other, more complex middle agents, such as subcontractors, are not available in this architecture.

Social system/dynamics

Social interactions are the main focus of the KaOS architecture, and it provides a rich and comprehensive mechanism for defining the ways in which agents interact.

The interaction model includes the speech (illocutionary) acts and the sequencing of the messages. The speech acts are similar to those of KQML (Finin *et al*., 1997), only they are more general, and hence KaOS could be considered as a communication meta-architecture. Speech acts of any agent communication language can be implemented within KaOS. Message sequences are organised in conversations, a pattern of messages transferred back and forth between (two) agents, which are modelled using state transition diagrams. Each one of these sequences is pre-defined, hence when designing a KaOS architecture the designer should elicit all possible interaction sequences.

This model assumes honest and consistent agents and does not deal with problems of fraud and trust or with problems of security.

Gaia – Wooldridge *et al.*

Gaia is a methodology for agent-oriented analysis and design (Wooldridge *et al*., 1999; 2000), and it can be used as a meta-level model. Gaia includes all the aspects that are important in describing agent societies (individual agent aspects as well as social aspects, static aspects as well as dynamic aspects), but only at a high level. The reasoning behind this approach is the desire to leave low-level design and implementation issues as open ended as possible, allowing the designer to choose the architecture and programming language. While this approach has the advantage of being general and implementation-independent, it does not provide the designer with all the necessary tools to analyse and design the system.

Gaia makes an important distinction between the analysis (dealing with *abstract* concepts) and the design (dealing with *concrete* concepts) processes, and provides several models to be used at each phase:

The analysis phase is described using the following models:

- Roles model, which is composed of several other models:
  - Permissions;
  - Responsibilities (safety properties and liveliness properties); and
  - Protocols.
- Interactions model.

The design phase is described using:

- Agent model;
- Services model; and
- Acquaintance model.

The analysis models should be elicited first, and then the design models can be derived from them, while adding more details. Some approaches to information systems analysis and design claim that the translation process from the analysis phase to the design phase should be automatic and should not provide for any degrees of freedom; Gaia does not provide such direct translation.

Another aspect of the design process, which is missing in Gaia as well as in most analysis and design methodologies, is specific directions on how to perform the analysis process, and how to answer questions such as "what entities in the domain should become agents?" and "what is the goal of each agent?"

Gaia is based on object-oriented concepts, but it adds the notion of virtual organisation of agents (somewhat similar to the ideas of COT – Computational Organisation Theory). The analysis and design process is similar to the process of constructing a society of agents, defining the role and capabilities of each individual agent, and defining the way the society of agents is structured.

Individual agent/static structure

Gaia provides no support for constructs describing the internal structure of the agent, hence it is not clear what mechanism is used to transform the agent's perceptions into actions. This part of the methodology is completely open ended, and leaves the designer absolute freedom in deciding what cognitive model to use.

Individual agent/dynamics

In the *services model*, which is defined during the design process, the functionality of each agent type is specified. This model includes a description of the inputs, outputs, pre-conditions and post-conditions for each service. The services are derived from the protocol model (where the functions and goals of each role are detailed), and the pre − and post-conditions are derived from the responsibilities of the role.

Gaia does not prescribe an implementation approach for the services.

Social system/static structure

The *roles model* is used in the analysis phase to describe the social structure of the agent society. An agent plays a role (many-to-many relationship), and each role is associated with *responsibilities* (the agent's functionality and its goals), *permissions* (the right associated with the role, mainly rights to use resources, which allow the agent to realise the responsibilities of the role) and *protocols* (the allowable types of interaction).

The notion of resource used here is very limited and includes only information and knowledge (which the agents have).

The *agent model* is used in the design phase to specify the hierarchy of agent classes. Inheritance is not part of this model and the model only relates roles to agent classes, where an agent class, or agent type, may include one or more roles. The agent model also describes the number of instantiations of each agent type, i.e. the actual number of agents from each class that will be implemented.

The possible communication links between agents are defined in the *acquaintance model*, which is basically a very simple graph linking agent types.

Social system/dynamics

The *interactions model* (analysis phase) is used to specify the possible interactions between roles. Each protocol is an institutionalised pattern of interaction, and is very schematic – it specifies only the

purpose, initiator, responder, inputs, outputs and processing of the conversation, but not the exact ordering of messages (which is defined at a later stage).

There is no model that describes the ordering of messages and the types of communication allowed, such as that available in methodologies that use speech act models.

Gaia does not deal with the complex behaviours that can arise in an agent society, and it provides no tools for expressing notions such as trust, fraud, commitment, and security.

Australian AI Institute (AAII) – Kinny *et al.*

The AAII methodology (Australian AI Institute; although not mentioned in the references, this is the name by which this methodology is mostly referred to) (Kinny *et al*., 1996; Georgeff & Rao, 1995) is based on the BDI (Beliefs, Desires, Intentions) approach for designing agent-oriented systems. AAII is not an analysis-and-design methodology, but only a design methodology, as it does not include a conceptual model of the organisation and its environment; it includes only concrete entities, which relate to data objects.

This methodology consists of two viewpoints. The external viewpoint describes the social system structure and dynamics. It includes an agent model (the static structure of the system) and an interaction model (the dynamics of the system). The external viewpoint is independent of any internal structure of the agent (the cognitive model) and independent of the communication mechanism.

The internal viewpoint is composed of three models: the belief model, the goal model and the plan model. These models specify how an agent perceives the environment and how he chooses his actions based on his perception.

The design of agents in this methodology is restricted to the BDI model. BDI agents are capable of rationalising about their actions and creating an optimised plan (based on their knowledge).

Individual agent/static structure

The cognitive model is specified in the internal viewpoint and it includes the following concepts: events the agent may perceive, actions the agent performs, beliefs the agent holds, goals the agent adopts, and plans, which give rise to the agent's intentions.

The belief model specifies the knowledge of the agent. The agent has beliefs about the environment, the agent's internal state and the actions the agent is capable of performing. Belief sets comprise the possible beliefs and their properties, and are described using the belief set diagrams. The goal model specifies the goal states: the goals an agent may adopt and the events the agent can respond to. The plan model includes a description of the plan set – plans that the agent may possibly employ – and is depicted using the plan diagram (which is an extension of a state chart).

Individual agent/dynamics

The functionality of the agent and the way in which the internal state of the agent may evolve over time are not part of the BDI methodology, since the methodology is restricted to one specific architecture – the BDI model. This model describes specifically how all of the concepts that are included in the agent's cognitive map interact and affect each other and the agent's planning mechanism. The designer has no freedom in defining this part of the model and hence it is not part of the methodology. This is both the major advantage and disadvantage of this methodology. It commits to a very detailed and clear internal architecture, which saves a lot of time and effort, but at the same time it restricts the designer to adopting the BDI approach.

This model does not include learning.

Social system/static structure

The structure of the system is specified in the agent model, which includes a description of the agent classes and their relations (similar to the roles concept in most other methodologies) and

the agent instances of each class and when they come into existence. These are depicted in the agent class and instances diagram.

There is no explicit description in the methodology of how to design the agent society and what sorts of agent (e.g. middle agents and mediators) will be implemented.

Other components of the system structure are included in the interaction model, such as the responsibilities of each agent class and the control relationships between classes.

Social system/dynamics

Social activity is described in the interaction model, where the syntax and semantics of the interactions are defined.

MaSE – Multi-agent Systems Engineering

MaSE is a methodology and a language for designing agent systems. It was first introduced at the Agent99 conference in Seattle, by Scott DeLoach (DeLoach, 1999; Wood & DeLoach, 2000).

The methodology includes four steps: domain-level design, agent-level design, component design and system design, to be followed in that order. It uses two languages, AgML (Agent Modelling Language), which is a graphical language, and AgDL (Agent Definition Language), to describe the system-level behaviour and to specify the internal behaviour of the agent respectively. AgML is used in the domain level design and the system design steps, while AgDL is used in the agent level design and component design steps. Each language contains a number of diagrams that describe different aspects of the system.

MaSE is a design-driven methodology. It does not include an analysis process, and thus it does not provide tools for capturing the organisational environment. However, it goes beyond the design to support some of the initial implementation process. The methodology is based on object-oriented techniques (mainly OMT and UML), and extends them by adding constructs to capture the specific behaviour of multi-agent systems. MaSE was developed to support formal system synthesis and it is based on first-order predicate logic.

Below we will try to classify the steps and the languages of MaSE according to the individual/system and static/dynamic categories, although MaSE does not follow this categorisation.

Individual agent/static structure

The agent's internal structure is defined in the third step – component design. There is no specific cognitive model that this methodology builds on, and the user is free to program any structure he may see fit.

Individual agent/dynamics

The agent behaviour is specified in the second step – agent-level design, using the AgDL language. This step builds on the agent conversation, which was identified in the previous step (domain-level design), and comprises the following steps: (1) Mapping actions identified in the conversations to internal components, (2) Defining data structures that were identified in the conversations (the data structures represent input and output from the agent) and (3) Defining additional data structures, internal to the agent.

There are no generic planning or scheduling architectures within the methodology, and users are free to implement any planning algorithm they may choose.

Social system/static structure

The system's structure and dynamics are specified by the AgML and its diagrams.

In the domain-level design, the agent types are identified and some of the system's dynamics are defined (see below). Three diagrams that are part of the AgML are used in the domain-level design:

agent diagram, communication hierarchy diagram and communication class diagrams. The agent diagrams describe the agent classes (including services and goals for each class) and their hierarchy.

Social system/dynamics

The system's dynamics are also specified using the AgML during the domain-level design and the system design steps. The agent diagram that is used in the domain-level design step describes, in addition to the agent classes, the possible coordination protocols (or conversations) between classes. Each conversation has a class hierarchy, which is specified in the communication hierarchy diagram. This diagram describes the relations between various conversations in the system. The communication class diagrams are a set of finite-state machines that define the states of each conversation.

MaSE provides a powerful tool for representing the agent's communication, but it lacks tools for capturing some of the more complex social interactions.

High-level and intermediate models – Elammari and Lalonde

Elammari and Lalonde introduced their agent-oriented methodology in the Agent99 conference at Seattle (Elammari & Lalonde, 1999). The methodology builds on two processes: high-level (the discovery phase) and intermediate models (the definition phase). The discovery phase deals with describing the organisational processes and workflow, and hence is very similar to our notion of an analysis phase. The authors do not distinguish between analysis and design phases and use the term agent in both the discovery and definition phases, but clearly the high-level model is used to capture the behaviour of the organisational environment where the system is to be implemented (and hence can be considered a conceptual model), and the intermediate models describe the architecture of the information system (and hence should be considered design models). The analysis in the discovery phase is done using UCMs (Use-Case Maps), which are very useful for visualising a workflow and work processes. The discovery phase provides a description of the scenarios, components, roles, scenario's pre- and post-conditions, and component's responsibilities and constraints.

The definition phase includes four models: the internal agent model, the relationships model, the conversation model and the contract model. These models are described below. The methodology includes guidelines on how to generate the models and how to answer questions such as "what entities should be represented by agents?" and "how to distinguish between the various agent types?" It also provides guidelines to define interrelations between the models.

The Elammari and Lalonde methodology is based on existing approaches in software engineering, and adjusts them to describe multi-agent systems.

Individual agent/static structure and dynamics

Both the agent's internal structure and the agent behaviour are represented in the internal agent model, which is derived from the high-level model. This model includes a description of the agent's goals (a desired state), pre-conditions (the beliefs that should hold in order for the goal to be executed), post-conditions (the effect of executing a goal on the agents beliefs) and tasks that are required to fulfil each goal.

Several plans can be used alternatively to realise a goal, and each plan can be specified using a finite-state machine.

The agents in this model are not able to make decisions or solve problems, nor can they learn from experience.

Social system/static structure

The relationship model specifies inter-agent dependencies (using the dependency diagram) and jurisdictional relationships (using the jurisdictional diagram). This model is also derived from the high-level model. The dependency diagram describes dependencies between service providers and

agents that require services. There are four types of dependency: goal, task, resource and negotiated dependency, and together these dependencies capture a large number of constraints and relationships frequently encountered in the business environment. The jurisdictional diagram specifies the authority hierarchy of agents. The hierarchies are based on roles and are used to allocate authorities and delegate policies.

Social system/dynamics

The conversation and contract models describe the system's behaviour. The conversation model identifies what messages are exchanged in order to fulfil the dependencies and jurisdictional relationships. This diagram lists all the messages an agent may receive and the possible responses to each one of these messages. The messages are based on speech acts.

The contract model specifies the obligations (commitments) and authorisations between agents about the services provided to each other. An organisation of agents is based on these commitments, and is used to facilitate cooperation and conflict resolution. A commitment means that the agent is willing to give access to its resources or services.

ALBERT and I* Languages – Yu *et al.*

Yu, Du Bois, Dubois and Mylopoulos describe in Yu *et al*. (1995) an analysis and design framework that includes two languages: I* and ALBERT.[19] I* was developed at the University of Toronto and ALBERT at the University of Namur. The authors did not explicitly distinguish the analysis and design phases. They did not relate the conceptual model to the design model, and in many cases there is a use of design entities in the analysis phase (see the "Account Handler" in the example provided in Yu *et al*. (1995)). Nevertheless, I* can be considered as an analysis language (referred to by the authors as an "understanding level" model), and ALBERT a design language (referred to by the authors as a "specification level" model). I* is used to "support the generation and evaluation of organizational alternatives" (Yu *et al*., 1995, p. 2) which is really the aim of the conceptual model – to present the current organisational structure and behaviour or future ones (while considering alternatives). The (Agent-oriented Language for Building and Eliciting Real-Time requirements) is used to "produce a requirements specification document for system developers" (Yu *et al*., 1995, p. 2). A limitation of this framework is the absence of an "automatic" transformation mechanism from the I* model to the ALBERT model. Another limitation is that no guidelines are provided on how to map organisational reality to the models and answer questions such as "what entities should be represented by agents?"

This framework is requirements-oriented, although it allows going back and forth between the conceptual and design models.

I* is a framework for modelling intentional relationships among strategic actors, and it is made up of two models: strategic dependency model and strategic rationale model. Both models deal with the dynamic nature of agent relations.

I* is represented in graphical and formal (using the Telos conceptual modelling language) forms. ALBERT supports the modelling of functional requirements by representing them as a society of agents, which interact in-order to fulfil an organisational need. ALBERT offers both a graphical representation and a textual one, which is based on logical formulae.

Individual agent/static structure

In I* (the conceptual model), the internal structure of each actor is modelled using the following constructs: beliefs, wants, abilities, goals, tasks, commitments and resource. There is no description in Yu *et al*. (1995) of the relations between these constructs.

[19] A more updated version of ALBERT (ALBERT II) is described in Du Bois (1995) and Schobbens and Petit (1999). We used the earlier version of ALBERT in this paper because it is integrated with I* for analysis and design.

In ALBERT (the design language), agents have states, which are a list of attributes and the values assigned to them (each attribute is specified as a table). States are used to describe the agent knowledge (of the external world and states of other agents). Actions are performed by agents in order to discharge contractual obligations, and change their state. Agents are not intentional and do not have goals.

Individual agent/dynamics

In I*, the strategic rationale model is used to describe the agent's behaviour. This model relates tasks, goals and resources in order to allow the agent to formulate a plan, and lists two types of relationship:

- means-end relationship: what other means exist for achieving the same task; and
- task decomposition: how a task can be decomposed to several sub-tasks.

In ALBERT (the design language), local constraints, which are defined as logical statements, define the admissible behaviour of the agents:

- effects on actions: the state changes that are performed by each action;
- causalities among actions: how the current state and an external action can trigger an agent's action;
- capability: same as above, only that the action is not triggered; it is made possible; and
- state behaviour: how state changes occur without actions (e.g. through time).

No mechanism is provided for allowing the agent to optimise its plans or for the agent to learn.

Social system/static structure

During the analysis phase the roles and positions of the agents are listed (in I*); however, there is no description of how the role and position hierarchies affect the agent's behaviour.

ALBERT does not include a description of the systems' structure.

Social system/dynamics

The second model in I*, the strategic dependency model, represents interactions between agents (or strategic actors) in terms of who depends on whom to perform what actions; it presents an external view of how agents depend on each other. The model includes four types of dependency link:

- task dependency: empowering other agents to perform a portion of a task;
- resource dependency: an agent (to perform a task) depends on the presence of another agent – the other agent is viewed as a resource;
- goal dependency: other agents bring conditions to enable reaching a goal; and
- softgoal dependency: same as goal dependency, but conditions are not defined strictly. The agent should consider the how conditions are met and decide accordingly.

The strength (status) of each dependency can be open, committed or critical.

In the design process (in ALBERT) interactions take two forms: (a) making information visible to other agents and (b) one agent's actions affect other agents. Constraints, too, have two types: local (described earlier) and cooperation constraint, limiting inter-agent interactions. The cooperation constraints are:

- action perception: how and under what conditions other agent's actions trigger own action;
- state perception: the ability to see (know) other agent's states;
- action information: what information about actions is made visible to other agents; and
- state information: what information about own states is made visible to other agents.

The description of the system's dynamics in I* and ALBERT is rich, but it does not include complex social interaction and notions such as negotiation and trust.

Agent-oriented methodology for enterprise modelling – Kendall *et al.*

Kendall *et al*. describe (1996a) steps towards constructing a methodology for the analysis (enterprise modelling) and design of agent-oriented information systems. The methodology is based on the integration of existing techniques, mainly the CIMOSA modelling framework for enterprise integration, the IDEF approach for workflow modelling, and the use-case-driven approach to object-oriented software engineering.

Kendall *et al*. do a good job of comparing these approaches, integrating them and adapting them to fit agent-oriented systems. The authors study the differences between objects and agents, and update existing methodologies to accommodate these differences.

The general framework for the design of the agents' cognitive map is based on the BDI approach, and agents are designed to reason and optimise their behaviours.

There is no explicit notion of time in this framework. This agent-oriented methodology provides a graphical representation of the system with no formal semantics.

Although this framework is said to support both the analysis and the design phases, there is no clear distinction between those phases and there are no well-defined guidelines on the A&D process and the use of all the diagrams.

Individual agent/static structure

An agent perceives the environment via sensors. The perceptions update the agent's knowledge about the world (his beliefs). An agent has goals, which represent states an agent wants to achieve. Based on these goals and his beliefs, he formulates a plan of actions. Plans are instantiated when an event occurs and conditions are met, and then it becomes an intention.

Individual agent/dynamics

The agent is able to reason and choose a plan, which turns into an intention. The agent's intentions include three types of task: (a) internal task that updates the agent's beliefs, (b) coordination task (interaction with other agents) and (c) invoking the effector, which impacts other objects. Beliefs change when a goal is achieved.

Sequence diagrams represent the conditions under which plans become intentions.

Use-case diagrams are used to represent the workflow processes. IDEF diagrams specify the reasoning mechanism and the consideration used by the agent to select a plan.

Social system/static structure

Use-case representation is a high-level diagram, which connects all the use cases and the agents. This diagram shows what use cases could be integrated into which agent's plan, and what use cases demand the cooperation of one or more agents. Also provided is a hierarchy of use cases that represents the hierarchy of work processes and a use-case inheritance diagram.

There is no description of other structures and relations in the system and, generally speaking, the agent's behaviour is not constrained by the system's structure.

Social system/dynamics

The agent uses its sensors to interact with various objects via messaging. The agents' interactions are modelled using diagrams, which represent the message workflow. However, there is no description of the type of message allowed (speech acts or others) or the message's structure. Coordination protocol diagrams are used to specify the various reaction options an agent has in each conversation.

It is worth mentioning that Kendall *et al*. extended this framework to create an architecture for designing multi-agent systems in Kendall *et al*. (1996b). The Layered Agent Pattern Language deals with reasoning agents, issues of cooperation and collaboration between agents, and mobility and

translation between semantics (extremely important in open systems). The architecture is made up of seven different layers (or models), which interact with each other and together form a comprehensive framework for designing agent systems. However, this language is not an analysis and design methodology, and therefore not included in this paper.

Agent-Oriented Analysis and Design (AOAD) – Burmeister

AOAD (Burmeister, 1996) is an analysis and design methodology that is a natural extension to existing object-oriented techniques. AOAD provides models to support the analysis phase (where entities in the domain are identified and modelled as agents) and the design phase (where the behaviour of agents is specified), but does not make a clear distinction between the two phases. AOAD is made up of three models: agent model, organisational model and cooperation model – all these models are used both to represent the organisation (i.e. it is a conceptual model, which is used during the analysis phase) and to describe the design of the information system (during the design phase).

The agent model is used to identify agents and their internal structure; the organisational model is used to describe the static structure of the system – the class hierarchies of agents and agents' roles; and the cooperation model represents the "dynamics in large" – the interactions among agents.

Individual agent/static structure

The agent's internal structure is described using the agent model. The model describes

- The agents and the environment (which entities are agents and which are a part of the environment). Although the model is basically a conceptual model, it is also used to describe design-level entities (e.g. using agents to realise the system's internal processes). There are no clear guidelines on how to determine which entity should be represented as an agent.
- The agents' motivations: the interests, preferences, responsibilities and goals of the agents, which drive their behaviour.
- The agents' knowledge and beliefs: knowledge about the external world and about other agents. This knowledge is used to execute the agents' plans (see below).

The AOAD approach is not committed to a specific cognitive architecture, and the designer is free to choose the attributes to describe the agents.

Individual agent/dynamics

The "dynamics in small" are also a part of the agent model. This part of the model defines the agent's behaviour via plans. Plans are series of actions and are activated either to fulfil a motivation or as a response to an external event. Plans can be described graphically using state transition diagrams.

Social system/static structure

The system's structure is defined by the organisational model, which includes the following steps:

- Identifying roles and responsibilities. Roles are mapped to agents, and influence the agent's motivation and thus its behaviour.
- Building inheritance hierarchy. Roles are classified into classes according to some attributes and agents with similar characteristics are grouped into one of these classes. This is very similar to OO techniques where classes are organised into a hierarchical structure.
- Structuring roles into organisations by decomposing the system into sub-systems.

As with the previous model, this model too can be used in both the analysis and design phases, which makes the development process more ambiguous and less clear.

Social system/dynamics

The system's dynamics are represented in the cooperation model using the following steps:

- Identifying cooperation and partners. Cooperation between agents can be around a goal fulfilment, by sharing resources or by synchronising actions. The cooperation type, the reason of cooperation, and the cooperating agents, are captured in this model.
- Identifying message types. The possible KQML message types to be used in the cooperation are captured in this model.
- Define cooperation protocols. Based on the two previous steps, the possible flows of messages among cooperating agents are captured in the cooperation protocols. Examples of such protocols are informing, querying and proposing.

MASB – Moulin & Brassard

MASB (Multi-Agent Scenario-Based) (Moulin & Brassard, 1996) is an analysis and design methodology, which is specifically designed to support the development of multi-agent systems. MASB makes a clear distinction between the analysis and design phases, and provides distinct models and graphical diagrams for each phase. MASB borrows from theatre and describes the reality using a metaphor of agents as characters playing roles in pre-defined scenarios. Complex behaviour of human and software agents is captured by scenarios (scripts) – a way of specifying the agent's social interactions, its behaviour, and the knowledge used to specify the behaviour. Agents have access to databases containing information that represents the world.

In the analysis phase the agent's behaviour is captured using scenarios. This phase is composed of five steps:

1. scenario description (using natural language),
2. role function description (using behaviour diagrams),
3. conceptual data modelling (using conceptual data and entity/object life cycle diagrams),
4. static and dynamic description of the world, and
5. system-user interaction modelling.

During the design phase, the scenarios are refined and formally specified as an agent's behaviour and knowledge structure that are used to implement the multi-agent system. This phase is composed of the following steps:

1. MAS architecture and scenario characterisation,
2. object modelling,
3. agent modelling,
4. conversation modelling, and
5. overall system design validation.

The first step of the analysis phase describes the organisational environment as scenarios, in a textual form. This description should emphasise roles played by agents, the information exchange, events that occur in the course of the scenarios, and actions taken by agents. An agent can play one or more roles, in one or more scenarios. Playing a role, an agent performs one or more activities.

The second step in the analysis phase, role function description, specifies the agent-roles relations, as well as roles-activities, activities-information and agent-environment relations. Behaviour diagrams are used to represent the scenarios as a tripartite graph whose nodes are processes (activities), accumulations (information) and environment (the world and other agents). Edges in the diagram are flows (process-environment link) and channels (processes-accumulations link).

These two phases capture the behaviour of the whole system (therefore cannot be classified into sub-categories) and are the starting point and the root of the analysis and design phases that follow. During the design phase the scenarios described in the analysis are used to generate design specifications, although in many cases these scenarios are further refined (e.g. by merging certain roles and splitting

others, introducing new agents and introducing detailed behaviours not considered previously). In the following, the analysis and design steps are detailed in the context of the representational sub-categories.

The notion of time is captured in MASB by temporal marks, which represent specific points in time or time intervals that may be significant for the agents. The agent has a clock and temporal marks are used to trigger actions.

Individual agent/static structure

The agent's knowledge is described in the behaviour diagrams as accumulations. During the third analysis phase, data conceptual modelling, the accumulations are further detailed in terms of their attributes and the "data conceptual structure" is defined (using a graphical representation). This model describes the main elements that will be included in the agent's database, and they can be represented using Entity-Relationship Diagrams (ERDs), object class diagrams, or "frames". At any given time, the agent's state is defined by the values associated with the attributes in the database. Another diagram, the entity (or object) life cycle diagram, is used to specify the allowed state transitions. An agent can deduce new knowledge (beliefs) from existing knowledge, based on a set of deduction rules (which are contained in a special segment of the agent's DB called the reasoning space).

In MASB, agents have the capabilities of intentional agents – they are able to record things as beliefs, make decisions by choosing goals to pursue, reason on their intentions and knowledge, and create plans of actions (aimed at achieving goals) and execute them. They do not, however, include an explicit model of other agent's beliefs, intentions, and plans (as social agents do).

The mental state of an agent is defined by its beliefs, goals, and expectations. Expectations are the agent's projection of future events, other agents' actions, or its own behaviour. Goals lead the agent's actions and are organised in a hierarchy of sub-goals.

During the design phase the agent's belief structures are defined, based on the conceptual data modelling (the first step of the design). The agent's knowledge is specified as belief structures or as objects shared by other agents and manipulated by the object server (see below).

Individual agent/dynamics

The agent's actions (processes) are associated with the roles the agent plays. Actions are specified by different types of transition rule, which have pre-conditions, post-conditions, and co-conditions. These conditions are composed of attributes of mental states and message structures. Actions are organised into actions plans, which are activated in order to achieve a goal.

The decision space is the decision structure composed of goal hierarchies and relations between goals and beliefs. The decision space is used in the process of decision-making and it includes different types of transition rules: activation rules (how a goal can activate its sub-goals), execution rules (conditions for execution of a goal), propagation rules (how the success or failure of a goal is propagated to other goals) and other rules that specify how a goal might be abandoned, suspended or resumed.

In the third step of the design phase, the decision spaces for each role are specified and detailed (using a graphical representation). The actions space, which is used to describe the plans used by the agents, is also specified (and represented graphically).

Social system/static structure

Agents are associated with roles, which are specified in the behaviour diagrams (second step of the analysis). Roles are associated with goals, plans and actions. MASB does not include additional hierarchies (other than role hierarchy) to describe the structure of the system.

The data structures that characterise the world are described in the fourth phase of the analysis – static and dynamic descriptions of the world – and represented using conceptual data models (the same

models used to describe the agent's knowledge). Object servers contain that knowledge, which is shared and manipulated by agents. The world is modelled as a set of objects (reactive agents) that have simple reaction plans and are capable of evolving. In order to receive information about the world, agents send requests to objects contained in the object server.

Other types of agent (aside from the intentional software agents and object servers) are the scenario manager (which manages all the scenarios and directs agents' actions at run time) and the conversation interpreter (which enables the user-agent interactions).

During the design (second step), the objects' structures are further specified, and the decision and action spaces of the conversation interpreter are detailed (fourth phase of the design – conversation modelling).

Social system/dynamics

Agents communicate by exchanging messages (certain messages correspond to speech acts), and messages are also used to model external events that happen in the world.

The fifth step of the analysis phase describes the interaction between users and software agents.

MASB does not deal with complex social behaviour (e.g. agents with contradictory goals) and its consequences (e.g. the need for a conflict resolution).

MAS-CommonKADS – Iglesias *et al*.

MAS-CommonKADS (Iglesias *et al*., 1996) is an analysis and design methodology for the development of multi-agent systems. It is, like CoMoMAS (Glaser, 1996), an extension of the knowledge engineering methodology CommonKADS (which was developed by the European Community's ESPIRIT program). Knowledge engineering (KE) methodologies are similar in many ways to A&D methodologies: they provide models for representing the (organisational) domain and for designing the software system. Techniques developed in KE can be easily used in the analysis and design of information systems, especially during the analysis phase, where knowledge about the static and dynamic nature of the environment is captured in a conceptual model. MAS-CommonKADS thus builds on the techniques and models of CommonKADS, and extends them by adding aspects that are relevant to multi-agent systems.

MAS-CommonKADS includes the following models to support the analysis phase:

- organisation model: the structure of the organisation,
- task model: the general tasks and processes that are performed in the organisation,
- agent model: the capabilities and characteristics of the agents,
- communications model: user-machine interfaces,
- coordination model: interactions between agents, and
- expertise model: the problem-solving knowledge used by an agent to perform a task.

All these models, except the coordination model (and some modifications to the agent model), are borrowed from CommonKADS.

MAS-CommonKADS includes one step prior to the construction of the analysis models – conceptualisation. During this phase early requirements are described using use cases (based on object-oriented techniques). Also included in the methodology are guidelines for determining what entities in the environment should be represented as agents.

During the design phase, the design model is constructed, consisting of the following models:

- application design: decomposition of the system into modules and choosing an agent architecture for each agent (deliberative, reactive or hybrid architecture); new types of agent may be introduced,
- architecture design: the exact types and numbers of the agents are determined and a specific multi-agent architecture is selected (this includes agent types, agents' knowledge and the use of ontologies, and coordination protocols to specify agents' communication), and
- platform design: decisions on software and hardware to run the application.

Individual agent/static structure and dynamics

The agent's structure and dynamics are represented in the agent model (analysis phase). This model represents services (facilities offered to other agents to help them satisfy their goals), goals (the objectives of the agents, which are satisfied by the reasoning mechanism), reasoning capabilities (different models can be used to represent reasoning), general capabilities (agents' skills and the communication language they understand) and constraints (norms, preferences and permissions).

The expertise model is used to represent the agent's knowledge in three sub-levels: domain knowledge, inference level (inference structures) and task level (ordering of the inferences).

Social system/static structure

The organisation model is used to describe the system's structure.

Social system/dynamics

The task model is used to represent general processes and tasks at the system level (before they are distributed to agents). The Coordination model is used to represent the agent interactions. These interactions are based on speech acts. The model contains of four elements: conversation (a set of interactions to ask for a service or request information from other agents), interaction (simple exchange of messages), capabilities (skills and knowledge of the participants in the conversation) and protocol (a set of rules that govern the conversation). Several graphical representations are used to represent the coordination model: message sequence charts (to represent scenarios identified using the use cases; an alternative representation is event trace diagrams), event flow diagrams (to model the generic behaviour of the agent and the knowledge exchange in interactions), and CEFSMs – Communicating Extended Finite State Machines (to model the control flow in interactions, including message events, external events and internal events).

CoMoMAS – Glaser

CoMoMAS (Glaser, 1996) is an environment to support the development of conceptual descriptions of multi-agent systems, and thus can be used as an A&D methodology. CoMoMAS includes tools and models to support the analysis (knowledge acquisition, modelling and representation) and design phases, as well as a mechanism for automatically generating code, based on the design model. CoMoMAS, like MAS-CommonKADS (Iglesias *et al*., 1996), is an extension of the knowledge engineering methodology CommonKADS (which was developed by the European Community's ESPIRIT program). Knowledge engineering methodologies are similar in many ways to A&D methodologies: they provide models for representing the (organisational) domain and for designing the software system. Techniques developed in KE can be easily used in the analysis and design of information systems, especially during the analysis phase, where knowledge about the static and dynamic nature of the environment is captured in a conceptual model. CoMoMAS thus builds on the techniques and models of CommonKADS, and extends them by adding aspects that are relevant to multi-agent systems and by providing an automatic code generator.

CoMoMAS uses the commercial product KADSTOOL for the knowledge acquisition and modelling. The result of these activities is a set of conceptual models that need to be operationalised. Formalisation of the analysis phase is obtained using the Conceptual Modelling Language (CML).

CoMoMAS is composed of four different modules to cover the whole knowledge engineering life-cycle: acquirer, modeller, constructor and coder. In this overview we will concentrate on the analysis and design activities realised by the first three modules.

The acquirer is used for knowledge acquisition, where knowledge is represented in the form of transcription protocols (graphical and textual) and a glossary. In the modeller, based on the protocols and the glossary, the CoMoMAS library is constructed. The library includes three types of knowledge category: domain (concepts, properties, relations and expressions), inference and task, which are

represented using domain hierarchies, semantic networks, inference structure and task hierarchies. Knowledge kernels (which include domain hierarchies, inference structures and task decomposition, and are partially translated into CML syntax) are then exported to the constructor. The constructor develops the design model and outputs the coder model sets. The coder then generates programming code, based on the model sets.

CoMoMAS includes all the models of CommonKADS (as described in the MAS-CommonKADS section): agent model, system (organisation) model, task model, expert (expertise) model, cooperation (communications) and design model.

Individual agent/static structure and dynamics

The agent's structure and dynamics are represented in the agent model (analysis phase). This model represents the agent's type, the agent's architecture, roles and communication protocols used by the agent. The expert model specifies the problem-solving techniques, cooperation methods, strategy and behaviours of the agent.

Social system/static structure

The system (organisation) model is used to describe the system's structure.

Social system/dynamics

The task model is used to represent general processes and tasks at the system level (before they are distributed to agents) and task hierarchies. The cooperation model is used to represent the agent interactions, and it contains descriptions of conflict resolution mechanisms, cooperation primitives and cooperation protocols.

Integrated Software Engineering (ISE) – Pont and Moreale

Integrated Software Engineering (ISE) (Pont & Moreale, 1996) is an integrated methodology for process-oriented, data-oriented, object-oriented and agent-oriented software development. The agent-oriented part is seen as the most general, thus we will describe only this portion of ISE. In general ISE is a high-level methodology, i.e. it describes the analysis and design process and provides some tools representing them, but does not contain explicit detailed procedures and models for describing the system.

ISE builds on well-established object-oriented techniques, integrates them and extends them to support the analysis (logical aspects – what the software is going to do) and design (physical model) of multi-agent systems. There is a well-defined distinction within ISE between the analysis and design phases. The analysis phase contains four steps:

1. the external agents: identification of agents in the system's environment,
2. the logical interactions diagrams: definitions of agents' conversations,
3. the system and external messages: the system's response to external messages, and
4. the message domain diagram (MDD): interaction between the agents.

The design phase includes the following steps:

1. the internal agent diagram: definition of goals and services of internal agents,
2. the class relationship diagram (CRD): relationships between agents, and
3. the physical interaction diagrams: description of processes.

Basically, ISE is focused on the system-level aspects, and the methodology is lacking tools for the representation of the agent's cognitive model and behaviour in the analysis phase. The description that follows is of the analysis and design steps.

Individual agent/static structure and dynamics

The agents' internal structure is not part of the analysis process (only for internal agents); it is only defined during the design phase. The first design phase, the internal agent diagrams, lists the agent's aims (goals) and the services they provide. Other than that the agent's structure and behaviour is completely open ended, and there is no cognitive architecture to guide the analysis and design of these aspects.

Social system/static structure

The external agents, their types and their goals are identified in the first step of the analysis process – the external agents. The hierarchy of the agent classes is represented by the class-relationship diagrams, as part of the second step of the analysis process – the logical interaction diagrams.

In the second design phase, the class-relationship diagram, the CRDs (borrowed from object-oriented methodologies) are constructed to represent the relationships between agent types.

The modelling of additional social structure (e.g. those based on roles or authority hierarchy) are not supported by ISE.

Social system/dynamics

In the second analysis phase, the interaction diagrams are constructed. These diagrams represent the system's behaviour, including the control and timing of processes. Interactions between agents are modelled as conversations. The logical interaction diagrams represent the type of message passed, the initiator and the respondent, and the temporal order of the messages.

In the third analysis phase, the system and external messages, key external messages to which the system must respond, are identified in the logical interaction diagrams.

Message Domain Diagrams (MDDs) are produced in the final analysis phase. These diagrams represent all the agents in the system and their interactions.

In the final design process, the physical interaction diagrams are defined based on the logical interaction diagrams. These diagrams define a series of methods (member function calls) – either in an implementation-independent fashion or in the form of code fragments.

Complex social interactions, which involve cooperation and collaboration between agents, are not considered in ISE.

AUML (the Agent UML) – Odell *et al.*

AUML – the Agent UML (Unified Modelling Language)[20] is an analysis and design methodology, which extends UML to represent agents (Odell *et al*., 2000). AUML builds heavily on the foundations of UML, both in models representing the organisational environment (conceptual models) and in models used to design the information system (which we called implementation models). Agents are seen as extension of active objects, exhibiting *dynamic autonomy* (proactive action capability) and *deterministic autonomy* (the autonomy to refuse an external request).

The aim of AUML is to provide both a formal and an intuitive semantics, through a user-friendly graphical notation, for the development of agent-oriented systems.

AUML extends the UML mostly in one aspect (or one representation sub-category) – it provides models to capture the dynamic aspect of inter-agent interactions. The methodology does not provide extensions to capture the cognitive map of the agent (individual/static structure), or the structural organisation of the agents (system/static structure).

AUML can serve to aid both the analysis and design processes, since it builds on existing models within UML, although the extensions proposed are mainly useful for the design process. The AUML extensions are made in the implementation model family (basically these are design models).

[20] Please distinguish AUML from UAML (Luck *et al*., 1997; Treur, 1999a; Treur, 1999c)

AUML provides extensions to the UML by adding a three-layer representation for Agent Interactions Protocols (AIPs), which define communication protocol as "an allowed sequence of messages between agents, and the constraints on the contents of these messages" (Odell *et al*., 2000). The authors chose to show how AIP could be used to adapt the UML to the agent paradigm, because agent interactions are a complicated enough subset of the representation model, and thus could show the way for the entire adaptation of UML to support the development of agent-oriented information systems.

The AIP levels are:

1. templates and packages to represent the protocol as a whole,
2. sequence, collaboration and activity diagrams to capture inter-agent dynamics and
3. the internal agent processing, modelled by activity diagrams and state charts.

Individual/static structure

AUML does not provide extensions to UML to represent the internal cognitive map of the agent: no additional representation models are provided to capture the agent's knowledge of the world and other components of its cognitive map, such as desires, intentions and mobility mechanisms. Nevertheless, within UML there are several models that can be used to capture these aspects, mainly the OML syntax.

Individual/dynamics

The third level of the AIP captures the internal agent processing, and thus provides a mechanism to represent the dynamics of the agent's cognitive map. AUML does not build on a specific cognitive architecture, and represents the agent's internal dynamics using activity diagrams and state charts.

System/static structure

AUML builds on the class hierarchy included in the UML model, and adds the notion of an agent playing a specific role. No other mechanism is provided to capture the system's structure.

System/dynamics

The first two levels of the AIP relate to the system's dynamics, and can be used both in the conceptual and in the design models. In level 1, templates and packages provide reusable solutions for message sequencing (these are part of UML). The authors advocate the use of a set of packages as an extension to UML, which will capture the specifics of agent interactions. The AIP packages serve as templates, which are parameterised model elements. Level 2 contains both sequence and collaboration diagrams, which represent the same semantics only in different graphical notation. Activity diagrams and state charts can also be included in level 2. Activity diagrams are used to represent processes, as they provide an explicit representation of the thread of control (which is useful in protocols that involve concurrent processing). State charts can be used to represent constraints for the protocol. Again, AUML includes extensions to UML's diagrams (for sequence, collaboration and activity diagrams) in order to model agent-based interaction protocols. The extensions include the use of communication acts and the support of concurrent threads of interactions.

AIPs can be specified in a detailed level, using a combination of diagrams (mainly activity diagrams), and thus capture the details of the interaction process. The more detailed the design models are, the easier it is to implement the systems.

Cassiopeia – Collinot *et al.*

Cassiopeia (Collinot *et al*., 1996) is a methodology for supporting the design of multi-agent systems. It focuses on system-level aspects (mainly on the collective behaviour of agents), and does not attend to designing concerns related to the agent's internal cognitive map.

Cassiopeia focuses on behaviour as a central thread, and the design process in Cassiopeia has three steps, related to three levels of behaviour:

1. elementary: the required elementary behaviours are defined, and based on them the agent types are defined,
2. relational: the structural description of the agents' organisation is defined (this is where relationships between behaviours are defined), and
3. organisational: the dynamics of agent interaction are described (organisational behaviour).

According to the Cassiopeia approach, functional dependencies are inherent to the collective achievement of a task. The whole set of these dependencies determine the coupling of the organisational problem. Two types of coupling are possible: (a) static – when there is no competition between agents and the designer can define in advance the structure, and (b) dynamic – when there is competition and the designer only defines possible organisational structures, which are instantiated within the problem-solving context.

Individual/static structure and dynamics

Cassiopeia does not provide a model to support the representation of either the cognitive components of the agent, or the dynamics internal to the agent.

System/static structure

Steps one and two of the design process describe the system's structure. In step one, the agent types are defined, based on a given analysis model. In step two, the coupling graph is produced, representing the dependencies between agents. These dependencies are termed influences. Then the relational behaviours that enable agents to identify and handle the influences are defined.

System/dynamics

In design step 3, the behaviours that will enable the agents to manage the formation, durability and dissolution of groups are defined. The occurrence of redundant groups indicates a redundancy of means to meet the need of a particular agent, named the trigger agent. As a way to control the formation of groups, the designer identifies the trigger agents (based on the influence graph) and determines selection methods for each. The designer defines the commitment signs that are produced by the trigger agents to indicate to other agents that a group is formed to meet their needs. The specification of these commitment signs defines the second group behaviour – joining behaviours. The last group behaviour to be defined is the dissolution behaviour.

Cassiopeia does not provide a specific mechanism for describing the agent communications and interactions.

MESSAGE/UML – Caire *et al.*

MESSAGE (Methodology for Engineering Systems of Software AGEnts) (Caire *et al*., 2001) is an agent-oriented SE methodology, aimed to support the analysis and design of multi-agent systems. The MESSAGE project is funded by Eurescom, a research organisation owned by European telecommunications companies. MESSAGE borrows from AUML (Odell *et al*., 2000) to include support for agents' interactions, from other agent-oriented approaches (namely Gaia and MAS-CommonKads), and from goal analysis techniques (Mylopoulos, 2001).

MESSAGE includes the following diagram types: organisation, goal, task, delegation, workflow, interaction and domain. The MESSAGE entity concepts are *ConcreteEntity*, *Activity* and *Mental-StateEntity.*

The main ConcreteEntities are: *agents* (have services and purpose), *organisation* (a group of agents working together towards a common purpose), *role* (a particular external characteristic of an agent in a particular context) and *resource* (non-autonomous entity used by agents).

The main Activity types are *task* (a task has pairs of situations describing pre- and post-conditions), *interaction* (borrows from the Gaia methodology; has purpose and participants) and *goal* (associates an agent with a desirable situation).

The analysis process in MESSAGE starts from a top-level decomposition, where the agent system is viewed as a set of organisations that interact with resources, actors and other organisations. At this level the focus is on identification of entities and their relationships. From the top level the analyst proceeds to a more detailed level of analysis, specifying the structure and the behaviours of the entities identified before.

Individual agent/static structure

Basically the internal structure of the agent is open ended, and can be based on one of several cognitive architectures. MESSAGE separates the agent's knowledge base from the inference mechanisms, but does not specify the exact type of knowledge in the knowledge base.

However, MESSAGE does prescribe several constructs related to the agent's internal structure, which are described in the *agent/role* view. An agent has a *role* (one or more), and may use *resources*. For each agent or role, the view describes what goals it is responsible for, what events it needs to sense, what resources it controls and what tasks it knows how to perform. The agent's motivation is captured by the *purpose* attribute. The agent has *goals* that are either derived from its purpose or are intrinsic to the agent's identity.

Individual agent/dynamics

An agent has *services* that describe its functional capabilities. Agent activities are tasks and interactions. An agent performs *tasks*, which are knowledge-level units of activity with pre- and post-conditions. Agent behaviour is specified by a set of behaviour rules.

Social system/static structure

The system's structure is defined in the *organisational view*, which describes agents, organisations, roles and resources, in one of several relationships: aggregation, power relationships (superior-subordinate relations) and acquaintance relationships.

The *goal/task* view describes goals, tasks and situations, and dependencies between them. They can be linked through logical dependencies to form graphs (i.e. describing goal decomposition and how tasks can be performed to achieve goals. Task decomposition is supported through causally linked sub-tasks.

Social system/dynamics

Interactions model system behaviour and agent coordination, and are described in the *interaction view*. Each interaction has more than one participant (initiator, collaborators and the motivator) and a purpose the participants are collectively trying to attain. An *interaction protocol* defines a pattern of *message* exchanges associated with an interaction. A message is an object communicated between agents, and it includes the following attributes: sender, receiver, speech act (specifying the sender's intent) and content.

Agent-Object Relationship (AOR) – Wagner

Agent-Object Relationship (AOR) (Wagner, 1999; 2000) is a design methodology for agent-systems. It also provides a description of how to transform these models into a database schema. The AOR architecture is inspired by Shoham's AOP (Shoham, 1993), and is an extension to OO methodologies.

Entities in AOR can be one of: object, agent, event, action, claim or commitment. Agent classes are associated with event, action, claim, or commitment. An organisation is viewed as a complex institutional agent, defining the rights and duties of its sub-agents.

Individual agent/static structure

AOR includes two basic types of entity: passive (objects) and active (agents). Similar to AOP, in AOR agents perceive the environment, and the cognitive state of agents is described in terms of beliefs, capabilities, choices and commitments.

Individual agent/dynamics

Agents can communicate, act, make commitments and satisfy choices. Agents send and receive messages, and communication in AOR takes the form of speech acts, according to a standard *agent communication language*.

Social system/static structure

There are three basic types of agent: artificial agents, human agents and institutional agents. Similar to OO modelling, agents and objects are organised in class hierarchies: specialisation and component. An organisation is an agent, composed of several subagents. The sub-agents have roles and positions. Duties to fulfil and rights to perform actions on behalf of the organisation are associated with agent roles.

Social system/dynamics

Agents' interaction is described in terms of message exchange. Agents can take actions and are committed to and have claims against other agents.

Open Distributed Application Construction (ODAC) – Gervais and Muscutariu

ODAC is an Architecture Description Language (ADL) for defining the design of agent systems, and it aims at filling the gap between the analysis and design phases. It supports the analysis, design and implementation phases. ODAC design is compliant with the OMG MASIF mobile agent platform, thus it is not implementation-independent. ODAC is based on the ISO Open Distributed Processing (ODP) standard that defines an architectural framework for the construction of distributed systems. ODAC is based on OO design methods, specifically on UML.

ODAC adopts the ODP reference model and defines five viewpoints to direct the development process. For the analysis phase: enterprise, information and computational views; for the design phase: engineering view; and for the implementation phase: technology view.

Individual agent/static structure and dynamics

Based on UML design; no specific constructs are described.

Social system/static structure and dynamics

In the enterprise, information and computational views the behaviour of the system is specified, as part of the analysis process. The objective of the system is detailed, as are the information that it handles and the tasks it carries out.

For the design model, the basic system-level concepts are role, community, objective, behaviour and action. Agents are grouped into communities, and each community has policies governing its

behaviour. An agent is associated with a role, and behaviour is associated with roles. Agents exhibit behaviour that is required to realise the objective of their community.

B&P – Bergenti and Poggi

The methodology presented by Bergenti and Poggi (2000) is an agent-oriented design methodology, and is implementation-independent. It is an extension to UML, which utilises UML's customisation capabilities to tailor the methodology to the AOIS domain. Agents are the atomic entities, and they interact with other agents through an agent communication language. Four agent diagrams are introduced:

- Agent system architecture, describing agent classes and their relations. Each class is characterised by the actions that an agent belonging to it can be requested to perform. Relations between classes may be used to express acquaintance relations.
- Role diagrams, specify the agent's role in interactions, i.e. the role that an agent plays in the protocol (see below).
- Protocol, modelling agents' communications over a set of roles.
- Ontology followed by agents, allowing definition of a model of the world composed of entities and relations. These relations are employed in agent communication to model the content messages that agents are allowed to use in communication.

These extensions to UML are focused on agents' communication in an open and heterogeneous environment.

Individual/static structure

Two elementary constructs are used to describe agent (class): responsibilities and messages. B&P does not specify any additional constructs for modelling the agent's cognitive map.

Individual/dynamics

In general, this aspect is open ended and no specific constructs are pre-specified. For each agent, a set of actions is defined. These actions are mostly intended to support agent communication.

System/static structure

The system's structure is defined by the set of agent classes and their relations. A specific relation is the acquaintance network, which is used to define which other agent each agent knows. Roles are associated with agents' classes, and are utilised in agent interactions.

System/dynamics

B&P extends UML by providing several diagrams for modelling agent communications. An agent's communication capabilities depend on the role it plays. Interaction protocols model the flow of the interaction and the message contents, and the ontology defines the semantics of each message.

Agent-Oriented Role-Based Workflow Modelling (AORBWM) – Yu and Schmid

AORBWM is a methodology for the analysis and design of agent systems, which borrows from business process modelling. It makes a clear distinction between the analysis and design phases, and provides guidelines to support the analysis and design processes. In AORBWM a business process is viewed as a collection of autonomous problem-solving agents, which interact with each other and have interdependencies. Agent classes are associated with *roles*, which are defined in terms of *goals*,

qualifications, *obligations*, *permissions* and *communication protocols*. Coordination of workflow is achieved through communication between agents.

AORBWM focuses on agents' system structure, and provides comprehensive models for capturing the complexities of agents' social structure.

Individual/static structure

Agents conduct conversations, perform activities, use resources and have capabilities. No constructs are specified for modelling the agents' cognitive states.

At the design level, there are three types of agent: personal agents, actor agents and internal agents.

Individual/dynamics

The actions the agent performs and the messages it exchanges are governed by the role it plays.

System/static structure

Agent classes are associated with *roles*, which define a prototypical function for an agent in a workflow. Roles are defined in terms of *goals*, *qualifications* (necessary pre-conditions for achieving the goals), *relationships* (with other roles in the workflow), *obligations* (the role's functionality: what activities the role must or must not perform), *permissions* (what activities the role is permitted to perform), *protocols* (the role plays a part in a set of protocols to achieve its goals) and *resource* (passively utilised during activity performance). *Relationships* among roles are categorised in two dimensions: *authority* (control relationships between two roles) and *cooperation* (other roles to cooperate with). One or more roles may be assigned to an agent. Agents are selected to play a role based on *organisational policies* and their *capabilities*.

Goals and activities can be decomposed, and are structured in a hierarchy.

System/dynamics

Interactions between roles fall into typical patterns of message exchange, which are called *protocols*. A protocol is specified by a set of rules governing the conversation among agents. *Conversation* is a set of interactions for requesting a service or information. An *interaction* is a simple interchange of messages, and carries the following attributes: *speech act*, *communication language*, *knowledge representation language*, *synchronisation*, *sender*, *receiver* and *ingredients*.

Appendix B: relevant work in neighbouring fields

In Section 5 we referred the reader to fields of research where we might search for models or theories that could be employed as foundations for modelling multi-agent systems, specifically to cognitive psychology, anthropology and organisational science.

However, there are other research areas where one might look for ideas about modelling agent systems. Several closely related fields that are interested mainly in designing systems will not provide full theories to support agent modelling, but might provide some useful insights into the constructs and relations required to model agents.

Researchers interested in developing agent-oriented A&D methodologies should look beyond the boundaries of this specific field to neighbouring fields that address similar problems. Below we review work in closely related research fields: (1) agent-oriented programming languages and control architectures, (2) agent-oriented simulation and automatic prototype generation frameworks, (3) DAI agent-oriented problem-solving frameworks and (4) Computational Organisational Theory (COT). We will shortly describe each neighbouring field, discuss its similarities to work in agent-oriented analysis and design, and point out how work in that field might be relevant to our discussion. When describing frameworks in neighbouring fields, we will concentrate only on the representation models used in these frameworks, and will ignore other features.

Agent-oriented programming languages and control architectures

In Abbott (1987), programs are described as a representation of the domain knowledge (plus control), and programming languages as modelling schemes to represent that knowledge. Hence implicit or explicit model schemes in agent-oriented programming languages may serve as a basis for the design models in agent-oriented A&D methodologies.

For example, the model of the agent's cognitive map within the programming language AOP (Shoham, 1993) contributes directly to our understanding of how to design agent systems (and, more specifically, how to design the agent's internal structure). The mental state of the agent in AOP is defined by the agent's beliefs (about the world, itself and other agents' mental states), capabilities (the ability to perform an action) and commitments (or obligations). Although AOP is a programming language, it is easy to see how ideas about modelling and representation of agents used in this framework are relevant to the design of agent-oriented information systems. An example of an agent-oriented control architecture is Concurrent METATEM (Fisher, 1996; 1999a; 1999b).

Agent-oriented simulation and automatic prototype generation frameworks

Techniques and tools under this category are not designed for solving problems, nor for creating a general framework for software engineering; they are designed for fast generation of multi-agent systems. Usually these multi-agent systems are used to simulate animal and human behaviours, to produce simple prototypes to test designs and architectures of agents, or even to develop full applications. Models within these frameworks focus mainly on design and implementation issues (and not on analysis issues), and hence provide representation that can be used to construct design models in A&D methodologies. Examples of frameworks in this field include DESIRE (Brazier *et al*., 1997a; Brazier *et al*., 1997b; Jonker & Treur, 1998; Research Group AI, Vrije University), ASE – Agent Simulation Environment (Luck *et al*., 1997), STEAM (Tambe, 1997) and SWARM (Minar *et al*., 1996).

DAI agent-oriented problem solving frameworks

DAI (Distributed Artificial Intelligence) problem-solving frameworks are used to design multi-agent systems for optimising some global goals. These systems are built by researchers in the field of AI, DAI and decision science, to solve problems such as planning and scheduling. What distinguishes methodologies in this group is that the models they build must be tractable and use formal tools.

Models within these frameworks focus mainly on design and implementation issues and not on analysis. Some of these design models can be employed during the analysis and design process. Examples of frameworks under this category are TAEMS (Decker, 1995; Prasaad *et al*., 1996), SIGAL (Maamar *et al*., 1997), and the "Language/Action Perspective" (Verharen & Weigard, 1994; Verharen *et al*., 1997).

Computational Organisation Theory (COT) frameworks

COT is a field of science that uses the power of computers to study and simulate human behaviour.[21] COT frameworks are greatly influenced by theories and models from the social sciences. These methodologies borrow models from psychology and the behavioural sciences to model the agents' internal processes, and ideas from organisational science to design the society of agents. Although COT frameworks are not generally thought of as analysis and design methodologies, they could easily be used to analyse any type of agent-oriented information system. These frameworks rely heavily on the agent paradigm and provide models to support the analysis process. The methodologies that belong to this group are very expressive and include complex and comprehensive models of the agent's cognitive behaviour and the behaviour of the agent society. Models of COT could easily be applied to the analysis phase (even if these models cannot be applied as-is, they provide insight into what constructs and relations are suitable for representing organisational processes). These models comply with many of the guidelines sketched in this paper; they are founded on psychological[22] and social theories, are requirements-driven and are comprehensive. Examples of COT frameworks are SDML (Carley & Gasser, 1999; Moss *et al*., 1998) and TOVE (Fox *et al*., 1998; Gruninger & Fox, 1994).

References

Abbott, R, 1987, "Knowledge abstraction" *Communications of the ACM 30* 8 664–671.

Agentlink, 2001, "Second meeting of the MSEAS Agentlink SIG" *AgentLink Special Interest Group on Methodologies and Software Engineering* http://polaris.ing.unimo.it/MSEAS/Amsterdam.html.

Arnold, P, Bodoff, S, Coleman, D, Gilchrist, H and Hayes F, 1991, "An evaluation of five object-oriented methods http://www.hpl.hp.com/techreports/91/HPL-91-52.html HP Labs.

Barber, K, Liu, T, Goel, A and Martin, C, 1999, "Conflict representation and classification in a domain-independent conflict management framework" *Proceedings of the Third International Conference on Autonomous Agents* 346–347.

Bergenti F and Poggi A, 2000, "Exploiting UML in the design of multi-agent systems" *Proceedings of the ECOOP – Workshop on Engineering Societies in the Agents' World 2000* (ESAW'00) 106–113 http://www.jfipa.org/publications/adr.php?l=ESAW2000Bergenti.

Bordini, R, 1999, *Contribution to an Anthropological Approach to the Cultural Adaptation of Migrant Agents*, PhD thesis. Department of Computer Science, University College.

Bradshaw, J, 1997, "An introduction to software agents" in J Bradshaw (ed.) *Software Agents* AAAI Press/The MIT Press.

Bradshaw, J, Dutfield, S, Benoit, P and Woolley, J, 1997, "KaOS: toward an industrial strength open agent architecture" in J Bradshaw (ed.) *Software Agents* AAAI Press/The MIT Press.

Brazier, F, Dunin-Keplicz, B, Treur, J and Verbrugge, R, 1997a, "Modelling internal dynamic behaviour of BDI agents" *Proceedings of the Third International Workshop on Formal Models of Agents* (MODELAGE'97) 36–56.

Brazier, F, Eck, P and Treur, J, 1997b, "Modelling a society of simple agents: from conceptual specification to experimentation" in R Conte, R Hegselmann and P Terna (eds.) *Simulating Social Phenomena, Proceedings of the International Conference on Computer Simulations and Social Sciences* (ICCSandSS'97) Springer-Verlag.

[21] A good introduction to COT can be found in Carley and Gasser (1999) and Epstein and Axtell (1996).

[22] The internal structure of the agent is usually based on a cognitive architecture. University of Michigan (1999) includes a comprehensive study of the theory in this research field, as well as description of specific architectures.

Brazier, F, Jonker, C and Treur, J, 1999, "Principles of compositional multi-agent systems development" *Proceedings of the IFIP World Computer Congress, Conference* (IT&KNOWS'98) http://www.cs.vu.nl/ treur/SIG1.jtreur.html.

Brinkkemper, S, Hong, S, Bulthuis, A and van den Goor, G, 1995, "Object-oriented analysis and design methods: a comparative review" http://wwwis.cs.utwente.nl:8080/dmrg/OODOC/oodoc/oo.html.

Briot, J and Gasser, L, 1998, "Agents and concurrent objects" *IEEE Concurrency* **6**(4) 74–77 http://www.computer.org/concurrency/pd1998/p4toc.htm.

Bubenko, J, 1993, "Extending the scope of information modelling" *Proceedings of the Fourth International Workshop on the Deductive Approach to Information Systems and Databases* (DAISD 1993) 73–97.

Burmeister, B, 1996, "Models and methodology for agent-oriented analysis and design" in K Fischer (ed.) *Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Artificial Intelligence* DFKI document D-96–06, 7–17 http://www.dfki.uni-kl.de/dfkidok/publications/D/96/06/abstract.html.

Caire G, Leal F, Chainho P, Evans R, Garijo F, Gomez J, Pavon J, Kearney P, Stark J and Massonet P, 2001, "Agent-oriented analysis using MESSAGE/UML" *The Second International Workshop on Agent-Oriented Software Engineering* (AOSE-2001) www.cediti.be/download/software_agent.pdf.

Carley, K and Gasser, L, 1999, "Computational organizational theory" in G Weiss G (ed.) *Multiagent Systems* The MIT Press.

Carley, K and Newell, A, 1994, "The nature of the social agent" *Journal of Mathematical Sociology* **19**(4) 221–262.

Ciancarini, P, Omicini, A and Zambonelli, F, 1999, "Coordination models for multi-agent systems" *AgentLink News* 3–6 http://www.agentlink.org/.

Collinot, A, Drogoul, A and Benhamou, P, 1996, "Agent-oriented design of a soccer robot team" *Proceedings of the Second International Conference on Multi-agent Systems* (ICMAS96) 41–47.

Coltell, O and Chalmeta, R, 1999, "CafeAGS: an architecture to support formal requirements specification for multi-agent system" *Proceedings of the Ninth European Workshop on Modelling Autonomous Agents in a Multi-Agent World* (MAAMAW99).

Decker, K, 1995, "Environment-centered analysis and design of coordination mechanism" Ph.D. thesis, University of Massachusetts, Amherst http://dis.cs.umass.edu/research/taems.html.

DeLoach, S, 1999, "Multiagent systems engineering: a methodology and language for designing agent systems" *Proceedings of the Agent-Oriented Information Systems (AOIS) in the Third International Conference on Autonomous Agents*.

Du Bois, P, 1995, "The ALBERT II language: on the design and the use of a formal specification language for requirements analysis" Ph.D. thesis, Institut d'informatique, FUNDP, Namur.

Elammari, M and Lalonde, W, 1999, "An agent-oriented methodology: high-level and intermediate models" *Proceedings of the Agent-Oriented Information Systems (AOIS) in the Third International Conference on Autonomous Agents*.

Epstein, J and Axtell, R, 1996, *Growing Artificial Societies* The Brookings Institution.

Ferber, J and Gutknecht, O, 1998, "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) 128–135.

Finin, T, Labrou, Y and Mayfield, J, 1997, "KQML as an agent communication language" in J Bradshaw (ed.) *Software Agents* AAAI Press/The MIT Press.

Fisher, M, 1996, "A brief introduction to METATEM" http://www.doc.mmu.ac.uk/STAFF/michael/cmet96/cmet96.html.

Fisher, M, 1999a, "A formal method for agent based systems – a position paper" http://www.cs.vu.nl/ treur/SIG1.mfisher.html.

Fisher, M, 1999b, "Representing abstract agent architectures" *Proceedings of the Fifth International Workshop on Agent Theories, Architectures and Languages* (ATAL-98) 227–241.

Fowler, M, 1992, "A Comparison of object-oriented analysis and design methods" in *Proceedings of the Seventh ACM Conference on Object-Oriented Programming Systems, Languages, and Applications* (OOPSLA' 92).

Fox, M, Barbiceanu, M, Gruninger, M and Lin, J, 1998, "An organizational ontology for enterprise modeling" in M Prietula, K Carley and L Gasser (eds.) *Simulating Organizations: Computational Models Of Institutions and Groups* AAAI Press/The MIT Press.

Franklin, S and Graesser, A, 1996, "Is it an agent, or just a program? A taxonomy for autonomous agents" in *Proceeding of the Third International Workshop on Agent Theories, Architectures and Languages* (ATAL-96) http://www.msci.memphis.edu/ franklin/AgentProg.html.

Genesereth, M, 1997, "An agent-based framework for interoperability" in J Bradshaw (ed.) *Software Agents* AAAI Press/The MIT Press.

Georgeff, M and Rao, A, 1995, "BDI agents: from theory to practice" *Proceedings of the first International Conference on Multi-Agent Systems* (ICMAS-95) 312–319.

Gervais MP and Muscutariu, F, 2001, "Towards an ADL for designing agent-based systems" *The Second International Workshop on Agent-Oriented Software Engineering* (AOSE-2001) 49–56 http://www-src.lip6.fr/homepages/Marie-Pierre.Gervais/AOSE2001.pdf.

Glaser, N, 1996, "Contribution to knowledge modelling in a multi-agent framework (the CoMoMAS approach)" Ph.D. thesis, L'Universtite Henri Poincare, Nancy http://www.loria.fr/ glaser/extern/these/these3.html.

Gruninger M and Fox, M, 1994, "An activity ontology for enterprise modelling" *Third IEEE Workshop on Enabling Technologies – Infrastructures for Collaborative Enterprises* http://www.eil.utoronto.ca/papers/model.html.

Iglesias C, Garijo M, Gonzalez J and Velasco J, 1996, "A methodological proposal for multiagent systems development extending CommonKADS" *Proceedings of the Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop* 25–1/17 http://ksi.cpsc.ucalgary.ca/KAW/KAW96/iglesias/Iglesias.html.

Iglesias C, Garijo M and Gonzalez J, 1999, "A survey of agent-oriented methodologies" *Proceedings of the Fifth International Workshop on Agent Theories, Architectures and Languages* (ATAL-98) 317–330.

Jennings NR, 2000, "On agent-based software engineering" *Artificial Intelligence* **117**(2), 277–296.

Jennings N, Sycara K and Wooldridge M, 1998, "A roadmap of agent research and development" *Autonomous Agents and Multi-agent Systems* **1** 275–306

Jonker C and Treur J, 1998, "Agent-based simulations of reactive, pro-active and social animal behaviour" in *Proceedings of the 11th International Conference on Industrial and Engineering Applications of AI and Expert Systems* (IEA/AIE'98) 584–595.

Kendall E, Malkoun, M and Jiang C, 1996, "A methodology for developing agent based systems for enterprise integration" in P Bernus and L Nemes (eds.) *Modelling and Methodologies for Enterprise Integration* Chapman and Hall.

Kinny D, Georgeff, M and Rao, A, 1996, "A Methodology and modelling technique for systems of BDI agents" *Agents Breaking Away: Proceeding of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World* (MAAMAW96) 56–71.

Klusch M, 1999a, "Preface" in M Klusch (ed.) *Intelligent Information Agents*, Springer.

Klusch M, 1999b, "Introduction" in M Klusch (ed.) *Intelligent Information Agents* Springer.

Lind, J, 2000, "Issues in agent-oriented software engineering" *The First International Workshop on Agent-Oriented Software Engineering* (AOSE-2000) 45–58.

Luck, M, Griffiths, N and d'Inverno, M, 1997, "From agent theory to agent construction: a case study" *Intelligent Agents III: Proceeding of the Third International Workshop on Agent Theories, Architectures and Languages* 49–64.

Luck, M, Coulter-Smith, E, Simon, E, Davidsson, P, Klusch, M, Moss, S, Roth, V, Sierra, C, Wooldridge, M, and Zambonelli, F, 2001, "AgentLink II: continuation of a network of excellence for agent-based computing" *AgentLink*, year-one report, IST-1999–29003 http://www.agentlink.org/admin/docs/2001/2001-021.pdf.

Maamar, Z, Moulin, B, Bedard, Y and Babin, G, 1997, "Software agent-oriented frameworks meet georeferenced digital library interoperability" *Digital Library Magazine*, September 1997 http://www.dlib.org/dlib/september97/laval/09laval.html.

Monarchi, D, Gretchen, I and Puhr, A, 1992, "Research typology for object-oriented analysis and design" *Communications of the ACM* **35**(9) 35–47

Moss, S, Gaylard, H, Wallis, S and Edmonds, B, 1998, "SDML: a multi-agent language for organizational modelling" *Computational and Mathematical Organization Theory* **4** 43–69 http://www.cpm.mmu.ac.uk/cpmrep16.html.

Moulin, B and Brassard, M, 1996, "A scenbario-based design method and an environment for the development of multiagent systems" *Proceedings of the First Australian workshop on Distributed Artificial Intelligence* 216–231

Mylopoulos, J, 1999, "Requirements-driven information system engineering" *Slides from the Agent-Oriented Information Systems (AOIS) in the Third International Conference on Autonomous Agents*.

Mylopoulos, J, Castro, J and Kolp, M, 2000, "Tropos: towards agent-oriented information systems engineering, position paper" *The Second International Bi-Conference Workshop on Agent-Oriented Information Systems* (AOIS2000).

Odell, J, Parunak, H and Bauer, B, 2000, "Representing agent interaction protocols in UML" *Proceedings of the AAAI Agents 2000 Conference* 121–140.

OMG, 1992, "Object Management Group, object analysis and design survey of methodologies" OMG document 92–10–3, Object Management Group (OMG), Inc.

Pont, M and Moreale, E, 1996, "Towards a practical methodology for agent-oriented software engineering with C++and Java" Technical Report 96–33, Leicester University, Department of Engineering, http://www.le.ac.uk/engineering/mjp9/lued9633.ps.

Prasaad, M, Decker, K, Garvey, A and Lesser, V, 1996, "Exploring organizational designs with TAEMS: a case study of distributed data processing" *Proceedings of the Second International Conference on Multi-agent Systems* (ICMAS96) 283–290 http://www.keihanna-plaza.co.jp/ICMAS96/.

Research Group Artificial Intelligence, Vrije University, Amsterdam, The DESIRE Research Programme, http://www.cs.vu.nl/vakgroepen/ai/projects/desire/desire.html.

Reticular Systems, 1999, Agent Construction Tools, http://www.agentbuilder.com/AgentTools/index.html.

Schobbens, P and Petit, M, 1999, "The ALBERT approach to Agent Systems Design" http://www.cs.vu.nl/ treur/SIG1.pyscobbens.html.

Shoham, Y, 1993, "Agent-oriented programming" *Artificial Intelligence* **60**(1) 51–92.

Sloman, A, 1999, "Are brains computers" http://www.cs.bham.ac.uk/ axs/.

Sloman, A and Logan, B, 1999, "Building cognitively rich agents using the SIM_AGENT toolkit" *Communications of the ACM* **42**(3) 71–77 ftp://ftp.cs.bham.ac.uk/pub/groups/cog_affect/0-INDEX.html#contents.

Tambe, M, 1997, "Agent architecture for flexible, practical teamwork" *Proceedings of the Fourteenth National Conference on Artificial Intelligence* (AAAI 97) 22–28 http://www.informatik.uni-trier.de/ley/db/conf/aaai/aaai97.html.

Treur, J, 1999a, "Report of the first SIG meeting" http://www.cs.vu.nl/ treur/SIG1report.html.

Treur, J, 1999b, "Methodologies and software engineering for agent systems – second meeting" *AgentLink News* **3** http://www.agentlink.org/.

Tveit, A, 2001, "A survey of agent-oriented software engineering" *Proceedings of the First NTNU CSGS Conference*, http://www.jfipa.org/publications/AgentOrientedSoftwareEngineering/.

University of Michigan, Department of Electrical Engineering and Computer Science, 1999, "A survey of cognitive and agent architectures" http://krusty.eecs.umich.edu/cogarch0/.

Verharen, E and Weigard, H, 1994, "Agent-oriented information systems design" *Poster Proceedings of the International Symposium on Methodologies for Intelligent Systems* (ISMIS'94) 378–392.

Verharen, E, Dignum, F, and Bos, S, 1997, "Implementation of a cooperative agent architecture based on the language-action perspective" *Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages* 31–44 www.citeseer.nj.nec.com/verharen97implementation.html.

Wagner, G, 1999, "Agent-oriented enterprise and business process modeling" *Proceedings of the First International Workshop on Enterprise Management and Resource Planning Systems* (EMRPS'99).

Wagner, G, 2000, "Agent-oriented analysis and design of organizational information systems" *Proceedings of Fourth IEEE International Baltic Workshop on Databases and Information Systems*.

Wand, Y and Weber, R, 1989, "An ontological evaluation of systems analysis and design methods" in ED Falkenberg and P Lindgreen (eds.) *Information Systems concepts: An In-Depth Analysis* Elsevier Science Publishers.

Wand, Y and Weber, R, 1990, "Mario Bunge's ontology as a formal foundation for information systems concepts" in P Weingartner and G Dorn (eds.) *Studies on Mario Bunge's Treatise* The Poznan Studies in the Philosophy of the Sciences and the Humanities.

Wand, Y and Weber, R, 1993, "On the ontological expressiveness of IS analysis and design grammars" *Journal of Information Systems* **3** 217–237

Wand, Y and Weber, R, 1995, "On the deep structure of information systems" *Information Systems Journal* **5** 203–223

Wand, Y and Woo, C, 1999, *Ontology-Based Rules for Object-Oriented Enterprise Modelling*, the University of British Columbia, June 1999.

Wickler, G, 1999, "Tools and software for building intelligent agents" http://afrodite.itc.it:1024/agents/tools.html.

Wood, MF and DeLoach, SA, 2000, "An overview of the multi-agent systems engineering methodology" *The First International Workshop on Agent-Oriented Software Engineering* (AOSE-2000).

Wooldridge, M and Jennings, R, 1995a, "Intelligent agents: theory and practice" *The Knowledge Engineering Review* **10**(2) 115–152.

Wooldridge, M and Jennings, R, 1995b, "Agent theories, architectures and languages" *Proceedings of the First International Conference on Multi-Agent Systems* (ICMAS-95).

Wooldridge, M and Jennings, R, 1998, "Pitfalls of agent-oriented development" *Proceedings of the Second International Conference on Autonomous Agents* (Agent 98) 385–391.

Wooldridge, M, Jennings, N and Kinny, D, 1999, "A methodology for agent-oriented analysis and design" *Proceedings of the Third International Conference on Autonomous Agents* 69–76.

Wooldridge, M, Jennings, N and Kinny, D, 2000, "The Gaia methodology for agent-oriented analysis and design" *Journal of Autonomous Agents and Multi-Agent Systems* **3**(3), 285–312.

Yu, E, 1995, "Modelling strategic relationships for process reengineering" Ph.D. thesis, Department of Computer Science, University of Toronto.

Yu, E, 1997, "Why agent-oriented requirements engineering" *Proceedings of the Third International Workshop on Requirements Engineering: Foundations for Software Quality* 171–183 http://www.cs.toronto.edu/eric/#Publications.

Yu, E, Du Bois, P, Dubois, E and Mylopoulos, J, 1995, "From organization models to system requirements: a 'cooperating agents' approach" *Proceedings of the Third International Conference on Cooperative Information Systems* (CoopIS-95), 194–204.

Yu, L and Schmid, B, 1999, "A conceptual framework for agent-oriented and role based workflow modeling" *Proceedings of the Agent-Oriented Information Systems (AOIS) in the Third International Conference on Autonomous Agents*.

Zambonelli, F, 2001, "The methodologies and software engineering for agent systems special interest group: introduction and report of first meeting, Prague, July 9–11" http://polaris.ing.unimo.it/MSEAS/PragueReport.html.