The Knowledge Engineering Review, Vol. 16:4, 349–373. © 2001, Cambridge University Press
DOI: 10.1017/S026988890100025X Printed in the United Kingdom

Agent orientation in software engineering*

GERHARD WEIß

Institut für Informatik, Technische Universität München, D-80290 München, Germany; e-mail: weissg@in.tum.de

Abstract

Agent-Oriented Software Engineering (AOSE) is rapidly emerging in response to urgent needs in both software engineering and agent-based computing. While these two disciplines coexisted without remarkable interaction until some years ago, today there is rich and fruitful interaction among them and various approaches are available that bring together techniques, concepts and ideas from both sides. This article offers a guide to the broad body of literature on AOSE. The guide, which is intended to be of value to both researchers and practitioners, is structured according to key issues and key topics that arise when dealing with AOSE: methods and frameworks for requirements engineering, analysis, design, and implementation; languages for programming, communication and coordination and ontology specification; and development tools and platforms.

1 Introduction

Recent developments in agent-based computing and software engineering have revealed a significant potential and urgent demand for a close interaction among these disciplines. On the one hand, as the number of fielded agent-based software systems grows it becomes important to have software engineering technology available that is specifically tailored for these systems. Thus software engineering is crucial to the industrial and commercial application success of agent-based computing. On the other hand, as today's and tomorrow's standard software systems are required to operate in increasingly complex – distributed, large, open, dynamic, unpredictable, heterogeneous and highly interactive – application environments, it appears to be very promising and natural to build these systems in terms of agent and multi-agent technology. Thus agent orientation can serve as an useful paradigm in software engineering. The field emerging as a result of this mutual demand for interaction has been referred to as Agent-Oriented Software Engineering (AOSE).

This article wants to support and guide both researchers and practitioners in navigating through and becoming acquainted with available literature on AOSE. Creating such a guide to AOSE is challenging for three major reasons. First, AOSE constitutes a very young field that has not yet settled on unique and commonly accepted criteria for evaluating methods, techniques and tools. As a response to this, not only pointers to completed and elaborated approaches were included in this guide, but also pointers to approaches in a relatively early and exploratory stage of development. Second, AOSE constitutes an interdisciplinary field. As a response to this, the guide also contains pointers to related work from the broader context of software engineering and agent-based computing. Third, AOSE is a field that evolves very rapidly. For that reason this guide cannot be guaranteed to provide pointers to all work relevant and related to AOSE. The guiding principle was to make this guide as comprehensive as possible while at the same time keeping it as focused as necessary.

* The author would like to thank the reviewers for their valuable comments. Parts of this article resulted from an evaluation study on AOSE which was conducted within a research project supported by Deutsche Forschungsgemeinschaft under contract Br609/11-1.

The article is structured as follows. Section 2 deals with various basic issues raised by AOSE. This section points to work on the general principles and ideas underlying AOSE (2.1), work on the relationships between agents and objects (2.2) and work contributing to an assessment of agent orientation (2.3). Section 3 treats AOSE-related methods and frameworks for requirements engineering.[1] There are two lines of approach within requirements engineering, known as agent-oriented requirements engineering and goal-oriented requirements engineering, that are of particular relevance from the point of view of AOSE; pointers to work from both lines are provided. Section 4 overviews methods and frameworks for analysis, design and implementation of agent-oriented software. This includes pointers to approaches primarily based on agent and multi-agent technology (4.1), object-oriented technology (4.2) and knowledge engineering technology (4.3). What is also offered are pointers to work on the formal specification and verification of agent-oriented software (4.4) and further notes on key issues of agent-oriented analysis and design (4.5). Section 5 concentrates on AOSE-relevant languages. Three types of language are distinguished: programming languages for implementing agent-oriented software (5.1), languages for specifying communication and coordination among agents (5.2) and languages for specifying ontologies that enable agents to share and reuse knowledge (5.3). Section 6 focuses on development tools and platforms for agent-oriented software, and points to research prototypes as well as commercial products. Section 7 provides an overview of further approaches that apply standard concepts and formalisms known from software engineering to agent-oriented software. The concepts and formalisms considered are design patterns, software architectures, use cases and scenarios and the Unified Modelling Language (UML). Finally, Section 8 summarises key aspects and identifies urgent open issues that need to be addressed to ensure a further successful development of AOSE.

2 Basic literature

2.1 Foundations

AOSE is concerned with the engineering of software that has the concept of agents as its core computational abstraction. There are several readable articles, in particular Jennings (2000) and Jennings and Wooldridge (2002), that treat various key aspects of AOSE and of the paradigms of agent-based computing in general. A recent useful overview of the state of the art in AOSE is Wooldridge and Ciancarini (2001). Earlier papers offering useful initial considerations on AOSE are Genesereth and Ketchpel (1994); Gustavsson (1994) and O'Hare and Wooldridge (1992). There are two collections of papers on AOSE (Ciancarini & Wooldridge, 2001; Wooldridge *et al.*, 2002), and related collections with a somewhat broader engineering perspective on agent systems are Garijo and Boman (1999) and Omicini *et al*. (2000). These collections are a good starting point for exploring the field with its various facets.

2.2 Agents and objects

Dealing with AOSE requires us to deal with the notion of agency. Many different perspectives of agency have been described and discussed, and there is nothing like a "universally accepted" definition of what exactly determines agenthood. Among the key texts that seek to contribute to a clarification of the concept of agents are Franklin and Graesser (1997) and Wooldridge and Jennings (1995). Examples of other readable introductory texts on software agents are Bradshaw (1997) and Nwana (1996). Well-written course-level texts on computational agency are Russell and Norvig (1995, Chapter 2) and Wooldridge (1999). Books that broadly cover agent and multi-agent technology are

[1] In the literature the terms method and framework are not used uniformly, and in this article both are used to refer to a structured description of steps, activities, and/or guidelines that aim at successfully realising one or several phases of the software life cycle.

Bradshaw (2002) and Weiß (1999). Despite dissension in detail, however, there is an increasingly broad agreement on the usefulness of characterisations of the following kind, adapted from Wooldridge and Jennings (1995b):

> An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives.

Although not explicitly expressed by this characterisation, there is also a broad agreement on the importance of considering multiple, *interacting* agents rather than a single agent acting in isolation. In fact, in most cases the terms agent-based system and multi-agent system are used synonymously.

There is an ongoing debate on the relationships between agents and objects. Valuable papers throwing light on this issue are Jennings (2000) and Odel (1999). Recent publications in the field seem to indicate that this debate converges to the following broad consensus:

- the concept of agents is significantly different from the concept of objects in that it allows for a qualitatively different perspective of complex systems and their development;[2] and
- there is room for both the agent concept and the object concept because they are concerned with different levels of computational abstraction.

In other words, there is a shift from an "agents *versus* objects" controversy toward an "agents *and* objects" perspective reflecting the growing agreement on the need for both agents and objects.

2.3 Aspects of assessing agent orientation

Benefits of agent orientation The question why it is worth taking an agent-oriented approach to software is addressed in various articles. Among them are, in particular, Jennings (2000) and Jennings & Wooldridge (2002). As argued there, a key point is that agent-oriented software is well suited for complex – open, networked, large and heterogeneous – application domains. This is mainly due to the fact that the agent concept, as a first-order abstraction, allows a very natural and intuitively clear modelling and implementation perspective. This application-oriented characterisation of the benefit of agent orientation is also emphasised in Weiß (2001) where it is argued that an application is particularly suited for an agent-oriented approach if it involves many components which

- are not all known a priori (i.e. at requirements-specification and design time),
- cannot all be assumed to be fully controllable in their behaviour (e.g. due to conflicting interests and non-public individual preferences) and
- must interact on a sophisticated level of communication and coordination to achieve their individual or joint design goals.

Domains in which such applications can be found range from electronic and mobile commerce, over supply-chain and business process management, to telecommunications and logistics. For a great part it is this advantage of agent orientation which has caused the broad and fast-growing interest in agent-oriented software and its engineering.

[2] The main reason for this is that objects only encapsulate identity ("who"), state ("what"), and passive behavior ("how, if invoked"), while agents additionally encapsulate multiple degrees of freedom in activity and interaction ("when," "why," "with whom" and "whether at all"). As a consequence, the concept of agents is much more adequate for capturing autonomous and flexible behaviour at the cognitive and the social level. These considerations could be refined by distinguishing between active and passive objects; the former have their own threads of control (resulting in a decoupling of method execution from method invocation), whereas the latter have not. Although active objects are much closer to agents than passive objects in that they also show some kind of autonomy (or, at least, independence of invocation), they still differ with regard to their behavioural freedom and flexibility. As stated in Wooldridge and Ciancarini (2001, p. 6), "active objects are essentially agents that do not necessarily have the ability to exhibit flexible autonomous behavior". This also indicates that in practice it cannot be expected that it is always easy to decide whether a real-world entity should be treated as an agent or as an object. Interestingly, available standard characterisations of both "agents" and "objects" are of quite general flavour, which often makes the identification of real-world representatives of both concepts difficult.

As discussed by Jennings (2000), agent orientation can be viewed as a natural next step in the evolution of a wide range of software engineering approaches. While this step does not supplant techniques such as object orientation, component ware and design patterns, it provides a useful higher level of computational abstraction. In Jennings (2000) it is also pointed out that agent orientation fully supports the three techniques which Booch (1994, pp. 16ff) identifies as being essential for coping with software technology, namely decomposition, abstraction (information chunking) and hierarchy identification. Moreover, as also argued there, agent orientation does not represent a radical departure from current software engineering thinking; instead, legacy software can be relatively easily incorporated in agent software. From these benefits of agent orientation, and from the broad range of potential applications, it was concluded that agent orientation does have the capacity to succeed as a mainstream software engineering paradigm.

No silver bullet There is no silver bullet in software engineering (Brooks, 1986). This holds for object orientation (see, e.g., Aksit & Bergmans (1992) and Fischer *et al.* (1995)) and also for agent orientation. A difficulty specific to agent orientation seems to lie in the term "agent" itself, as it inherently tends to evoke mental associations and images (especially in the heads of people not familiar with agent technology) that are far from any software-technical relevance and realisation. In order to cope with this difficulty a developer always has to keep in mind that "agent" is to be used strictly as a technical term that must not be confused with the term "agent" as used in everyday life. Another difficulty of agent orientation results from its emphasis on autonomy. As pointed out in Jennings (2000), not handling autonomy carefully enough can lead to an unpredictability of the patterns and outcomes of interactions, and thus to undesirable emergent phenomena at the overall system level. It is further argued there that this difficulty can be avoided, or at least reduced, by applying rigid interaction protocols and preset organisational structures. Of course, applying such protocols and structures tends to limit agent autonomy and the potential advantages implied by it (e.g. self-organisation and robustness), and so further research is necessary to find better, more sensitive solutions to this difficulty.

Pitfalls and challenges A readable paper on pitfalls of the agent-oriented approach is Wooldridge and Jennings (1998). Examples of the 24 identified pitfalls are the following:

- You oversell agents.
- You see agents everywhere.
- You get religious or dogmatic about agents.
- You do not know what your agents are good for.
- You confuse buzzwords with concepts.
- You do not exploit related technology.
- You forget you are developing software.

The reader interested in useful broader considerations on the challenges raised by agent-based systems from the software engineering perspective is particularly referred to Crabtree (1998); Luck (1999); Petrie (2001) and Fisher *et al*. (1997).

Work identifying questions that matter Two useful papers that highlight basic questions one has to deal with when developing industrial agent-based systems are Parunak (1999) and Parunak (2000). Three examples of such questions are:

- What in a system becomes an agent?
- What communication channels and protocols do agents use?
- How mature is the application?

Another key question a developer has to deal with is what (multi-)agent architecture should be choosen for a given application. To answer this question is not trivial, given the many available architectures proposed in the literature Müller (1996). Helpful guidelines for answering this question can be found in Müller (1999).

A comprehensive list of general research questions on various aspects of multi-agent systems is given in Decker *et al.* (1989). These are not "software engineering questions", but they help to gain an intuitive understanding of the broad variety of issues and challenges one may be confronted with when building a concrete agent-based software system.

3 Methods and frameworks for requirements engineering

Requirements engineering is concerned with eliciting, modelling and analysing the functional and non-functional capabilities a software system must possess in response to real-world constraints. As noted in Nuseibeh and Easterbrook (2000), requirements engineering is often regarded as the front-end activity in the software development process, although it plays an important role in the management of change in all phases of software development. There are two closely related streams of work on requirements engineering, known as agent-oriented and goal-oriented requirements engineering, that are of particular relevance from the point of view of AOSE.

Agent-oriented requirements engineering The importance of considering *active* software and environmental components was realised in the field of requirements engineering more than a decade ago. The seminal paper in this respect is Feather (1987), in which a simple framework for modelling agents and for reasoning about their behavioural choice and constraints was introduced. Today the modelling of agents is considered as a particular area of concern in requirements engineering (Lamsweerde, 2000), and so it is not surprising that there is an increasing number of requirements engineering approaches that primarily rely on the concept of agents. Together these approaches establish a line of research and application which is usually referred to as *agent-oriented requirement engineering* (Yu, 1997). Two readable papers that clarify the role of the concept of agents in requirements engineering are Yu (2001) and Yu (2002). As pointed out there, it is important to distinguish between two notions of agents: agents as concrete software artifacts (which is the predominant notion in current AOSE), and agents as conceptual modelling constructs (which is the predominant notion in requirements engineering). As it is further noted in these two papers, there is no one-to-one correspondence between these two notions – in particular, an agent-as-modelling-construct may eventually not be materialised as an agent-as-software-artefact. Based on these considerations it is proposed to distinguish two conceptions of AOSE, namely "engineering of agent-oriented software" (EAOS) and "agent-oriented engineering of software" (AOES).

Key examples of requirements engineering approaches that take the concept of agents as the primary guiding concept are the following:

- ***i**** (Yu, 1997a). ***i**** – this naming refers to the notion of distributed intentionality – is a modelling framework whose central construct is that of an agent having intentional properties such as goals and commitments.
- ALBERT (Agent-oriented Language for Building and Eliciting Real-Time requirements) (Dubois *et al.*, 1994) and ALBERTII (Du Bois, 1997). These are formal, agent-centered requirements specification languages.

A good example of how ***i**** and ALBERTII can be combined is described in Yu *et al.* (1995). A complete requirements-driven framework for agent-oriented systems development that adopts ***i**** is

- Tropos (Mylopoulos & Castro, 2000) (Tropos stands for "easily changeable" and "easily adaptable", derived from the Greek "tropé").

Goal-oriented requirements engineering Closely related to agent-oriented requirements is what has been called *goal-oriented requirements engineering* in the literature (especially see Mylopoulos *et al.* (1999) and Yu & Mylopoulos (1998)). What makes goal-oriented requirements engineering frameworks attractive is that they are not restricted to functional requirements ("*what* the software is expected to do") but explicitly capture *non*-functional requirements (whose identification requires one to repeatedly ask goal-directed questions like *why*, *how*, and *how else*). Such non-functional requirements, sometimes also called quality requirements or soft(-goal) requirements, refer to qualities

like responsibilities, environmental interactions, reliability, flexibility, integrity and adaptability, and thus to qualities that also play an essential part in the realm of agent-based systems. Goal orientation in requirements engineering supports an explicit identification and evaluation of goal alternatives. Two key examples of complementary goal-oriented frameworks are

- KAOS (Knowledge Acquistion in autOmated Specification) (Dardenne *et al*., 1993) and
- NFR (Non-Functional Requirements) (Chung *et al*., 2000).

While KAOS is a formal framework having its focus on requirements acquisition, NFR is a qualitative framework having its focus on the representation of and reasoning about non-functional requirements. Within the NFR framework the concept of softgoals – goals having no clear-cut definition and/or satisfaction criteria – is used to represent non-functional requirements.

It is finally mentioned that there is no sharp borderline between agent-oriented and goal-oriented requirements engineering. For instance, in ***i**** goals are always associated with agents, while ALBERT allows one to talk about agents without talking about goals and NFR primarily deals with goals but not with agents. In other words, goal orientation and agent orientation in current requirements engineering approaches neither include nor exclude each other.

4 Methods and frameworks for analysis, design and implementation

This section overviews available methods and frameworks for the analysis, design and implementation of agent-oriented software. There is a wide variety of such methods and frameworks, and the criterion applied here to characterise and structure this variety is the disciplinary background on which the different approaches are based. This criterion is particularly appealing because it reveals the main foci, intentions and principles underlying the different methods and frameworks. The following disciplinary backgrounds can be distinguished:

- *Agent and multi-agent technology* Approaches having this background are characterised by a clear focus on capturing social-level abstractions such as agent, group or organisation, that is, on abstractions that are above the conventional object level. These approaches are treated in Section 4.1.
- *Object orientation* Approaches with this background are characterised by the attempt to appropriately extend existing object-oriented techniques such that they also capture the notion of agency. These approaches are considered in Section 4.2.
- *Knowledge engineering* Approaches with this background are characterised by an emphasis on the identification, acquisition and modelling of knowledge to be used by the agent components of a software system. Approaches of this type are subject of Section 4.3.

Formal approaches to the specification and verification of agent systems are listed in Section 4.4, and further methodological considerations of more general flavour are provided in Section 4.5.

4.1 Approaches based on agent and multi-agent technology

There are a number of "pure" methods and frameworks having agent and multi-agent technology as their primary background. A key assumption underlying these approaches is that a grounding in object orientation falls short in the sense that it does not allow one to capture elementary characteristics of agency. Among the most often cited representatives of this type of approach are the following:

- Gaia (Generic Architecture for Information Availability) (Wooldridge *et al*., 2000). This is a method that distinguishes between analysis and design and associates different models with these two phases. Gaia focuses on organisational aspects in terms of concepts such as roles, interactions and acquaintances. See also Zambonelli et al. (2000), which discusses an extension of Gaia based on the concept of coordination models known from the area of standard coordination languages (see Section 5.2).

- SODA (Societies in Open and Distributed Agent spaces) (Omicini, 2001). This is another good example of an analysis and design method that concentrates on the social (inter-agent) aspects of agent systems and employs the concept of coordination models.
- Cassiopeia (Drogoul & Collinot, 1998). This is a design method that distinguishes three levels of behaviour – elementary, relational and organisational – and aims at capturing both structural and dynamic aspects of the target system.
- AALAADIN (Ferber & Guytknecht, 1998). This is a general analysis and design framework that has its focus on the organisational level of multi-agent systems and is built on the three core concepts of agents, groups and roles. A formal extension of this approach towards specifying the requirements on the overall multi-agent system dynamics is proposed in Ferber *et al*. (2000).

Apart from these best-known "pure" approaches, recently several other distinct approaches have been proposed that are mainly concerned with issues above the standard object level. Four good examples are the following:

- EXPAND (Expectation-oriented analysis and design) (Brauer *et al*., 2002). This is an analysis and design method that emphasises the aspect of autonomy and introduces expectations held by individual agents as a first-order abstraction.
- In Dignum *et al*. (2002) an analysis and design method is described that emphasises the distinction between markets, networks and hierarchies as agent society frameworks.
- In Elammari and Lalonde (1999) a method is proposed that distinguishes between two phases – discovery and definition – within which several models of the agents' external and internal behaviour are generated. The method deals with both the visualisation and specification of behaviour and employs use case maps.
- In Zambonelli *et al*. (2001) it is argued that organisational abstractions – organisational rules, organisational structures and organisational patterns – should play a central role in the analysis and design of multi-agent systems. Initial considerations on an organisation-oriented methodology based on these abstractions are provided.

Another elaborated agent-oriented method is Tropos (see Section 3); the distinguishing feature of Tropos is that it covers requirements analysis as well.

4.2 Approaches based on object-oriented technology

There are several approaches to agent-oriented software systems that have the object-oriented paradigm (e.g. Booch (1994)) as their starting point. Prototypical examples of such approaches are the following:

- KGR (Kinny *et al*., 1996). This is a design and specification method for a particular class of agents, namely BDI agents (Georgeff & Rao, 1995). KGR extends the Object Modelling Technique (OMT) and considers two viewpoints – external and internal – of agent systems.
- MaSE (Multi-agent Systems Engineering) (Wood & DeLoach, 2001). This method covers design and initial implementation through two languages called AgML (Agent Modelling Language) and AgDL (Agent Definition Language) and builds upon OMT and UML.
- MASSIVE (Multi-agent SystemS Iterative View Engineering) (Lind, 2001a). This method covers analysis, design and code generation, and combines standard software engineering techniques such as multiview modelling, round-trip engineering and iterative enhancement.
- AOAD (Agent-Oriented Analysis and Design) (Burmeister, 1996). This analysis and design method proposes the use of extended class responsibility cards (CRCs) and the use of both the Object Modelling Technique (OMT) and the Responsibility Driven Design (RDD) method known from object-oriented development.
- MASB (Multi-Agent Scenario-Based) (Moulin & Brassad, 1996). MASB is an analysis and design method that covers issues of both objects and agents via behaviour diagrams, data models, transition diagrams and object life cycles.

Good examples of work that do not describe complete methods but extensions of concepts and techniques well known in the realm of object-oriented software engineering are the following:

- In Kendall (1998) it is proposed to use extended role models known from object-oriented software engineering as abstractions and patterns for agent-oriented analysis and design.
- AOR (Agent-Object-Relationship modelling/diagrams) (Wagner, 2000). AOR is an agent-oriented extension of the conventional Entity-Relationship modelling technique.

Examples of other approaches that aim at extending the object-oriented perspective to cope with issues of agency are Bryson and McGonigle (1998); Gadomski (1993); Pont and Moreale (1996) and Satapathy and Kumara (1999).

4.3 Approaches based on knowledge engineering technology

Knowledge engineering (e.g. Studer *et al*., 1998)) has served as another fruitful background for new agent-oriented development methods and frameworks. The two best examples of such methods currently available are the following:

- CoMoMAS (Conceptual Modelling of Multi-Agent Systems) (Glaser, 1996). This is an elaborated extension of the CommonKADS methodology (Schreiber *et al*., 1999), supporting analysis, design and automated code generation. CoMoMAS focuses on knowledge engineering issues arising in multi-agent contexts and integrates a commercial tool called KADSTOOL for the conception of expertise models.
- MAS-CommonKADS (Multi-Agent System CommonKADS) (Iglesias *et al*., 1998). This is another extension of CommonKADS that supports analysis and design of agent-oriented systems. MAS-CommonKADS adds object-oriented methods such as the Object Modelling Technique (OMT), Object-Oriented Software Engineering (OOSE) and Responsibility Driven Design (RDD) as well as protocol engineering methods such as the Specification and Description Language (SDL) and Message Sequence Charts (MSCs).

Another example of work aiming at an extension of CommonKADS to meet social level requirements is Gustavsson (1998). By its very nature knowledge engineering has a very close relationship with the engineering of agent and multi-agent systems (more specifically, of knowledge bases used by agents). For that reason it appears to be promising to think about development approaches based on established knowledge engineering methods such as MIKE or PROTEGE.

4.4 Formal specification and verification

Generally, specification is concerned with the functionality of the desired system, given the customers' expectations and needs; the key question to be addressed is what product should be built. Verification is concerned with the correctness of the product, given its specification; the key question to be addressed is whether the product is built correctly. A readable overview of formal specification and verification approaches for agent systems is Wooldridge (1998). As noted there, the use of logics appears to be a very successful way to a formal specification of agent systems. Most prominent examples of logical specification frameworks are the following:

- the theory of intention (Cohen & Levesque, 1990);
- the belief-desire-intention model (Rao & Georgeff, 1995); and
- $\mathcal{LORA}$ (Logic of Rational Agents) (Wooldridge, 2000).

Another logic-based specification scheme, allowing for a declarative representation of multi-agent systems, is described in Singh *et al*. (1993). A well-known alternative formal specification approach is

- DESIRE (DEsign and Specification of Interacting REasoning components) (Brazier *et al*., 1997).

This approach has its roots in knowledge engineering, although its specifications and their semantics can be formalised via temporal logics. DESIRE differs from other formal knowledge engineering specification approaches (see van Eck *et al*. (2001) for an overview) in that it maintains several local states rather than a single global one – this makes DESIRE particularly interesting from the agent systems perspective. A pioneering approach to the automated compilation of agent specifications is described in Rosenschein and Kaelbling (1986).

With respect to verification, two types of main approach can be distinguished:

- axiomatic approaches, that is, approaches based on techniques of theorem proving (see Fisher & Wooldridge (1997); Wooldridge (1992) and Schild (1999) for good examples); and
- semantic approaches based on model checking (see Benerecetti *et al*. (1999) and Rao & Georgeff (1993)).

Some available approaches to AOSE employ standard formal methods, techniques and languages that are well known in software engineering. Among these formalisms are, in particular,

- the state-based language *Z* (Spivey, 1992) and
- Petri net theory.

A good example of a *Z*-based formal framework for agent system specification is described in d'Inverno and Luck (2001). More general considerations on the value of *Z* for agent-based system specification are provided in Fisher (1997). Good examples of Petri net-based specifications of multi-agent systems can be found in Holvoet (1996) and Moldt and Wienberg (1997). There is other work in mainstream computer science appears to be of relevance for multi-agent system specification and verification. This includes, for instance, reactive systems theory (Manna & Pnueli, 1995 and Pnueli, 1986) and concurrency theory (CCS, CSP, π-calculus, etc., see Roscoe (1997)). The full value of these theories in multi-agent contexts still needs to be explored.

Almost all methods and frameworks mentioned throughout this paper explicitly deal, at different levels and with different intensity, with coordination and communication. Some of them (e.g. MaSE (Wood & DeLoach, 2001)) use formal or semi-formal standard notations to describe and to represent coordination and communication among agents, while others (e.g. Gaia (Wooldridge *et al*., 2000)) rely on more informal ones. Examples of such standard notations are UML-type sequence diagrams and Petri nets. An alternative representation formalism, originally developed and used in linguistics and discourse analysis, are Dooley graphs (Dooley, 1976). For instance, these graphs have been used for the explication of relationships within agent-agent conversations (Parunak, 1996) and for the engineering of multi-agent coordination requirements (Singh, 2000).

The reader interested in a broader discussion of the use of formalisms for multi-agent systems is pointed to d'Inverno *et al*. (1997).

4.5 Further key issues

Hybrid approaches The three types of method and framework that can be distinguished by applying the disciplinary background as the characterising criterion are not fully orthogonal. For instance, an approach primarily based on knowledge engineering may show specific features of object orientation, and an approach based on multi-agent technology may also cover critical requirements engineering issues. Hence, rather than thinking of sharp borderlines between these types, one should assume a gradual transition among them. A good example of a method that explicitly attempts to integrate concepts and techniques from different backgrounds is

- MESSAGE (Methodology for Engineering Systems of Software Agents) (EURESCOM-MESSAGE, 2000).

This method, which covers all phases of software development, merges ideas, techniques and approaches such as KAOS-based requirements engineering, UML, CommonKADS, Gaia and the Rational Unified Process model. MESSAGE has been developed in response to the needs of the

telecommunications industry, but is applicable to other domains as well. Another, yet much less detailed and very general, analysis and design method which is intended to be extensible through techniques and concepts from different disciplines is the $\mathcal{AWIC}$ (Agents-World-Interoperability-Coordination) method (Müller, 1996).

Alternative characterisations The disciplinary background is, of course, not the only (though a very useful) criterion for characterising available methods and frameworks for agent-oriented software development. Five examples of alternative criteria are sketched below. None of them should be considered "the best" because each reveals relevant properties from a different perspective. At the moment no unified and generally accepted characterisation scheme is available, but it is obvious that ideally such a common scheme combines all five criteria in one way or another.

Perhaps the most obvious alternative criterion is the portion of the development process covered by a method. Most available approaches deal with analysis and design (e.g. SODA (Omicini, 2001)), although there are approaches that cover implementation as well (e.g. MASSIVE (Lind, 2001)).

Another alternative criterion is the *modelling level* on which a method primarily focuses. Based on this criterion, one can distinguish between approaches emphasising the intra-agent level (which concerns e.g. the individual agent's components, knowledge structures and reasoning strategies), the inter-agent level (which concerns e.g. communication and coordination protocols), and the supra-agent level (which concerns e.g. organisational structures, norms and social laws). As a rough indication it can be said that most available approaches primarily based on object-oriented technology tend to stress the intra-agent level, while available approaches primarily based on agent and multi-agent technology typlically emphasise the intra-agent and supra-agent levels.

A third, related criterion is the *developmental direction*, resulting in the distinction between bottom-up (e.g. AOAD (Burmeister, 1996)) and top-down (e.g. Aalaadin (Ferber & Gutknecht, 1998)) approaches. The former start by identifying and specifying intra-agent characteristics, whereas the latter start by identifying properties at the supra-agent level.

A fourth alternative criterion is *generality*. For instance, there are approaches which are less general in that they are designed to support specific agent architectures such as contract-net architectures (e.g. Cassiopeia (Drogoul & Collinot, 1998)) or BDI architectures (e.g. KGR (Kinny *et al*., 1996)), while other approaches are more general and independent of specific agent views (e.g. Gaia (Wooldridge *et al*., 2000)). This criterion could be further refined by distinguishing "technological generality" and "application generality".

Finally, a fifth alternative criterion is the *level of granularity*, that is, the level of detail considered by the approaches. This criterion could be further refined by considering granularity with regard to the different modelling levels mentioned above and/or with regard to the different phases of the software life cycle.

Available surveys Two useful surveys of available development methods and frameworks for agent-oriented software systems are available:

- The comprehensive survey in Arazy and Woo (2000) compares a number of development approaches, as well as several programming languages and simulation environments, with regard to their coverage of life cycle phases and their abstraction granularity.
- The survey offered in Iglesias *et al*. (1999) covers several agent-oriented extensions of object-oriented and knowledge engineering methods, and additionally points to several formal specification methods.

5 Languages

This section overviews languages for programming agent-based systems (Section 5.1), languages for communication and coordination among agents (Section 5.2), and languages for specifying ontologies (Section 5.3).

5.1 Programming languages

Most agent systems are probably written in Java and C/C++. Apart from these standard languages, several prototype languages for implementing agent-based systems have been proposed that all aim at enabling a programmer to better realise agent-specific conceptions. Taking a look at these languages also helps to understand the ideas behind and the challenges of programming agent-oriented systems. Three paradigms for implementing agent systems have been proposed: *agent-oriented programming* (Shoham, 1993) and, more recently, *market-oriented programming* (Wellman, 1996) and *interaction-oriented programming* (Singh, 1996; Huhns, 2001). The basic idea behind market-oriented programming is to view, design and implement agent systems according to economic principles of markets and market price systems. Against that, interaction-oriented programming is based on the idea that a designer's and programmer's focus should be on the events and processes occuring between (rather than within) agents. The formulation and elaboration of the paradigms of market orientation and interaction has just started, so in the following the focus will be on the agent-oriented programming paradigm.

Among the most prominent and best understood prototype languages following the agent-oriented paradigm are the following:

- AGENT-0 (Shoham, 1993) realises the basic ideas of the agent-oriented programming paradigm as formulated by Shoham. A language that extends AGENT-0 towards planning is PLACA (Thomas, 1995), and a language that aims at integrating AGENT-0 and KQML (see Section 5.2) is AGENT-K (Davies & Edwards, 1994).
- Concurrent MetateM (Fisher, 1995) allows to specify the intended behaviour of an agent based on temporal logics. A comparison of Concurrent MetateM and DESIRE (see Section 4.4) is presented in Mulder *et al*. (1998).
- AgentSpeak(L) (Rao, 1996) is a rule-based language that has formal operational semantics and that assumes agents to consist of intentions, beliefs, recorded events and plan rules. AgentSpeak(L) is based on an abstraction of the PRS architecture (Georgeff & Lansky, 1987). A formal specification of AgentSpeak(L) based on *Z* is presented in d'Inverno and Luck (1998).
- 3APL (Hindriks *et al*., 1999) incorporates features from imperative and logic programming. 3APL has well-defined operational semantics and supports monitoring and revising of agent goals. Work relating 3APL and AgentSpeak(L) is described in Hindriks *et al*. (1998).
- ConGolog (De Giacomo *et al*., 2000) is a concurrent logic-based language initially designed for high-level robot programming. Work relating ConGolog and 3APL is presented in Hindriks *et al*. (2000).

Other examples of languages following the agent-oriented programming paradigm are April (Agent PRocess Interaction Language) (McCabe & Clark, 1995), MAIL/MAI2L (Multi-agent Interaction and Implementation Language) (Steiner, 1996) and VIVA (Wagner, 1996). While standard concurrent programming languages do not support high-level agent modelling, most available languages for agent-based programming do not support concurrency (but see Concurrent MetateM and ConGolog). A system called DAISY that aims at overcoming this problem by including both an object-oriented language called CUBL (Concurrent Unit Based Language) and an agent-oriented language called MAPL (Multiple Agent Programmer Language) is described in Poggi (1995). The first commercially available language for the network-independent implementation of mobile agents is Telescript (White, 1997). A discussion of design choices for agent-oriented languages and their effects on programming open systems can be found in Burkhard (1995).

5.2 Languages for communication and coordination

The difficulty precisely handling coordination and communication increases with the sise of the agent-based software to be developed. In response to this a number of languages for coordination and communication have been proposed. The most prominent examples of such languages are the following:

- KQML (Knowledge Query and Manipulation Language) (Finin *et al*., 1997; KQML, 1999) is perhaps the most widely used agent communication language. An integration of KQML into Tcl/Tk is proposed in Cost *et al*., 1998).
- ARCOL (ARtimis COmmunication Language) (Sadek, 1991) is the communication language used in the ARTIMIS system (Sadek *et al*., 1997). ARCOL has a smaller set of communication primitives than KQML, but these can be composed.
- FIPA-ACL (FIPA Agent Communication Language) (FIPA, 1999) is an agent communication language that is largely influenced by ARCOL. Together FIPA-ACL, ARCOL and KQML establish a quasi standard for agent communication languages.
- KIF (Knowledge Interchange Format) (Genesereth & Fikes, 1992; KIF, 1999). This logic-based language has been designed to express any kind of knowledge and meta-knowledge. KIF is a language for *content* communication, whereas languages like KQML/ARCOL/FIPA-ACL are for *intention* communication.
- COOL (domain independent COOrdination Language) (Barbuceanu & Fox, 1996). COOL aims at explicitly representing and applying coordination knowledge for multi-agent systems and focuses on rule-based conversation management. Languages like COOL can be thought of as supporting a coordination/communication (or "protocol-sensitive") layer above intention communication.

Apart from these most prominent languages, several others showing unique properties have been proposed, for instance:

- ICL (Interagent Communication Language) (Martin *et al*., 1999) is a language that encompasses both agent-agent and agent-human communication and deals with conversational protocols and content descriptions.
- AgentTalk (Kuwabara *et al*., 1995) is a coordination protocol description language for multi-agent systems. AgentTalk supports the application-specific, incremental definition and customisation of coordination protocols.
- CoLa (Communication and coordination Language) (Verharen *et al*., 1998) allows for the specification of obligations and authorisations and supports the separation of tasks and contracts.
- TuCSoN (Tuple Centres Spread over Networks) (Omicini & Zambonelli, 1999) is a coordination model based on the notion of programmable communication abstractions called tuple centres. This model has its focus on Internet applications for mobile agents. An extension of TuCSoN towards security and topology is proposed in Cremonini *et al*. (1999).
- LuCe (Denti & Omicini, 1999) is a coordination language that is based on first-order logic and adopts tuple centres as coordination media. The semantics of LuCe (or LuCe -like languages) is described in Omicini (2000).
- STL++(Simple Thread Language ++) (Schumacher *et al*., 1999) is a language that provides a framework for describing the organisational structure of a multigent system. STL++ supports peer-to-peer, multicast and generative communication.
- SDML (Strictly Declarative Modelling Language) (Moss *et al*., 1996) is a language designed to facilitate modelling of multi-agent interactions.

A principal problem with available communication languages lies in the definition of a unique semantics – even implementations of quasi standard languages such as KQML resulted in different dialects that prohibit communication beyond proprietary multi-agent systems. This problem is addressed in a number of publications; good examples are the following:

- Singh (1998) proposes to emphasising social interaction rather than mental agency in the formal semantics of communication languages in order to avoid the problem of multiple dialects;
- a method for designing application-specific communication languages for which it is easier to verify semantic compliance is introduced in Pitt and Mamdani (1999); and
- the problem of determining conformity to the semantics by an independent observer is formally investigated in Wooldridge (1998).

A brief overview and discussion of main topics of interest in agent communication research can be found in Dignum (2000).

Here are pointers to some survey articles:

- A recent useful survey of intentional agent communication languages can be found in Kone *et al.* (2000).
- There are several readable introductions to and reviews of coordination/communication languages and models for parallel and distributed programming; see Arbab *et al.* (1998); Ciancarini (1996); Gelernter and Carriero (1992) and Papadopoulos and Arbab (1998). Though these languages and models do not explicitly deal with coordination and communication in agent systems, they obviously are relevant to it.

A valuable book bringing together a number of contributions centered around the coordination of agents on the Internet is Omicini *et al.* (2000).

5.3 Ontology specification languages

In order to increase interoperability and to enable agents to act jointly – to solve problems, to plan and to learn together rather than in isolation – especially in large-scale and/or open applications, it is necessary to specify a common ontology (i.e. an explicit and precise description of domain concepts and relationships among them) on the basis of which they can share and reuse knowledge. Several prototypical languages have been proposed that support the creation and editing of ontologies. Among the most elaborated examples of such languages are the following:

- Frame-based languages such as Ontolingua (Gruber, 1992) and Frame Logic (Kifer *et al.*, 1995). Both Ontolingua and Frame Logic extend first-order predicate logics. The key modelling primitive of these languages is frames, as known from artificial intelligence.
- Description logics such as CLASSIC (Bordiga *et al.*, 1989) and LOOM (MacGregor, 1994) which allow an intensional definition of concepts.
- CycL (Lenat & Guha, 1990), which extends first-order predicate logic and was developed to enable the specification of large common-sense ontologies.

As the number of agent-oriented Web-based applications (including all kinds of application requiring the processing of information on the Web as well as all kinds of e-commerce and B2B applications) increases, it becomes more and more important to have ontology specification languages that are conform to syntactic and semantic Web standards. The most prominent approaches to such languages are the following:

- SHOE (Simple HTML Ontology Extension) (SHOE, 2001) is a language that slightly extends HTML and enables a hierachical classification of HTML documents and the specification of relationships among them.
- XOL (Ontology Exchange Language) (XOL, 2001) is an XML − and frame-based language for the exchange of ontologies.
- OIL (Ontology Inference Layer) (OIL, 2001) aims at unifying formal semantics as offered by description logics, rich modelling primitives as offered by frame-based languages, and the XML and RDF web standards.[3] OIL can be seen as an extension of XOL offering both an XML-based and an RDF-based syntax.
- The DAML (DARPA Agent Markup Language) languages DAML-ONT and DAML-OIL (DAML-LANGUAGE, 2001). DAML-OIL, which replaces DAML-ONT and represents the state of the art in the field, has well-defined model-theoretic and axiomatic semantics.

[3] RDF (Resource Description Framework) is an XML-based framework for machine-understandable descriptions of Web resources of any type. For information on XML and RDF see `http://www.w3.org/XML/` and `http://www.w3.org/TR/rdf-schema/`.

Several editors for ontology creation and maintenance have been proposed. Three good examples of such editors are

- Protégé (PROTEGE, 2001), which supports single-user ontology acquisition,
- Webonto (WEBONTO, 2001), which supports multiple-user ontology acquisition over the Web, and
- OntoEdit (ONTOEDIT, 2001), which supports multilingual development of ontologies and multiple inheritance.

A very useful and broader introduction to ontologies is Fensel (2001). Readers interested in good introductory articles on ontologies are pointed to Gruber (1995) and Uschold & Gruninger (1996).

6 Development tools and platforms

A number of tools and platforms are available that support activities or phases of the process of agent-oriented software development. Most of them are built on top of and integrated with Java. While almost all available tools and platforms have their focus on implementation support, some of them do also support analysis, design and test/debugging activities. It is beyond the scope of this article to describe and compare the available tools and platforms in detail. However, in the following some of the most prominent representatives are listed. Examples of often cited *academic and research prototypes* are the following:

- ZEUS (ZEUS, 1999) is a toolkit that has been developed at the British Telecom Intelligent System Research Lab. A visualisation tool for agent applications built with ZEUS (or other toolkits) is described in Ndumu *et al*. (1999).
- JADE (Java Agent DEvelopment framework) (JADE, 1999) has been developed at the University of Parma, Italy.
- LEAP (Lightweight Extensible Agent Platform) (LEAP, 2000) is intended to be executable on small devises such as PDAs or phones. LEAP is being developed within Europe's Fifth Framework programme by several industrial and academic contract partners (Motorola, ADAC, BROADCOM, BT, Siemens and the University of Parma).
- AgenTool (AgenTool, 2000) is a Java-based graphical development environment that supports the MaSE method (see Section 4.2). AgenTool was originally developed at the Artificial Intelligence Lab of the Air Force Institute of Technology, Ohio.
- RETSINA (RETSINA, 2000) is a complex environment for networked intelligent agents that includes different (multi-)agent architectures, location and discovery services, middle agents and configuration management support. RETSINA has been developed at Carnegie Mellon University.
- JATLite (Java Agent Template, Lite) (JATLite, 2000), which has been developed at the Stanford Center for Design, is a package of Java programs that allows one to create software agents that communicate over the Internet.
- FIPA-OS (FIPA-OS, 2000) is a component-based toolkit for the development of FIPA-compliant agents. Two types of FIPA-OS are available, namely "standard" for execution on standard computers and "micro" for execution on PDAs.
- MADKIT (MADKIT, 1999) is a platform which is being developed at LIRMM (France). MADKIT is based on the Aalaadin model (see Section 4.1).

Other examples are SIM_AGENT (SIMAGENT, 1996), JAFMAS (Java-based Agent Framework for Multi-Agent Systems) (JAFMAS, 2000), ABS (Agent Building Shell) (ABS, 1999) which employs the language COOL (Section 5.2), OAA (Open Agent Architecture) (OAA, 1999), and Agentis (Kinny, 1999), which is a modelling framework for BDI agents.

Here are representative examples of *commercial products*[4] for developmental support:

[4] See the companies' web pages for free downloads and/or evaluation versions.

- AgentBuilder (AgentBuilder, 1999) is a tool offered by Reticular Systems Inc., USA. AgentBuilder is available in two versions: AgentBuilder Lite (entry-level) and AgentBuilder Pro.
- JACK (Busetta *et al*., 1999; JACK, 1998) is a commercial agent framework by Agent Oriented Software Pty. Ltd., Melbourne, Australia. JACK is oriented towards BDI agents.
- Intelligent Agent Factory (Intelligent Agent Factory, 2000) by Bits & Pixels, Texas, USA.
- Grasshopper (Grasshopper, 1998) is an advanced development platform for mobile agents launched by IKV++, Germany.

Related tools can be found at the IBM Aglets Development Kit homepage (IBM Aglets Development Kit, 2000) and the Microsoft Agent homepage (Microsoft Agent, 2000).

A comparison of four available platforms (AgentBuilder, JACK, MADKIT and ZEUS) can be found in Ricordel and Demazeau (2000). Some of the above-mentioned platforms (e.g. JADE, ZEUS, FIPA-OS, LEAP and Grasshopper) conform to the FIPA specifications (FIPA, 1999); Grasshopper is also compliant with the OMG MASIF (Mobile Agent System Interoperability Facility) standard (OMG MASIF Standard, 1999).

There are many testbeds available for agent-based systems. Most of them have a research-oriented focus on experimentation and exploration. An overview of older testbeds of that kind can be found in Decker (1996), and more recent examples are IMPACT (Subrahmanian *et al*., 2000), SWARM (Swarm, 2000), and Agent Factory (O'Hare, 1996). Good examples of testbeds explicitly oriented towards industrial needs and real-world applications are ARCHON (Cockburn & Jennings, 1996) and MECCA (Steiner, 1996).

7 Other approaches at the intersection of agent systems and software engineering

The complexity of modern software and software environments has resulted in an increasing use of concepts and formalisms that aim at building applications more efficiently and cost-effectively. Standard examples of such concepts and formalisms are design patterns, software architectures, use cases and scenarios and UML.[5] In the following, selected pointers to related work on agent system engineering are provided.

Design patterns (e.g. Gamma *et al*. (1995)) are abstract and reusable desciptions of solutions to particular (software) design problems. Good examples of work on design patterns for agent software are given below.

- In Deugo *et al*. (2000) a generic agent pattern format and specific patterns for agent-agent coordination are proposed.
- In Tahara *et al*. (1999) several patterns for agent behaviour (hence also called behaviour patterns), together with a development method using these patterns, are described.
- In Silva and Delgado (1998) an agent design pattern for developing dynamic and distributed applications is introduced.
- In Kendall and Malkoun (1996) and Kendall *et al*. (1997) a general pattern composed of seven layers (e.g. "sensory", "beliefs", "reasoning") for intelligent and mobile agents, together with layer-internal patterns, are proposed.
- In Aridor and Lange (1998) and Deugo *et al*. (1999) several patterns for mobile agents applications are identified.

Software architectures (e.g. Buschmann *et al*. (1996) and Shaw & Garlan (1996)) are structured descriptions of elements from which software systems are built. As mentioned earlier in Section 2.3, a number of agent architectures have been developed so far. Good examples of work dealing with agent architectures from an explicit software engineering perspective are the following:

[5] Another example is *Z*; see Section 4.4.

- In Kolp *et al*. (2001b) and Kolp *et al*. (2001a) several high-level organisational patterns for multi-agent architectures are presented.
- In Horn *et al*. (1998) and Reinke (2000) several object-oriented reference architectures for software agent applications are described.

The former work has a focus on requirements engineering issues, whereas the latter concentrates on design issues.

Use cases and scenarios play an important role in standard software and knowledge engineering (e.g. Erdmann & Studer (1998), Jacobson *et al*. (1994) and Weidenhaupt *et al*. (1998)). Several of the agent-oriented methods mentioned earlier apply use cases and scenarios during analysis and design; for instance, see Elammari and Lalonde (1999) (Section 4.1) and Wood and DeLoach (2001) and Moulin and Brassad (1996) (Section 4.2). Other good examples of related work on agent system engineering are the following:

- In Buhr *et al*. (1998b) and Buhr *et al*. (1998a) use case maps are applied to model and visualise overall system behaviour patterns.
- In Kendall *et al*. (1997) an object-oriented analysis method based on use cases and IDEF functions is described.

UML (Unified Modelling Language, e.g. Larman (1997)) is a *de facto* standard representational formalism in object-oriented analysis and design. Many of the methods mentioned in Section 4 apply UML to describe agent structures and interaction types. Here are good examples of work relating agency and UML:

- In Bauer (2002), Bauer *et al*. (2001) and Odell *et al*. (2001) an agent-oriented extension of UML, called AGENT UML or AUML, is described and illustrated. AUML is a result of cooperation between FIPA and OMG with the goal of increasing industrial acceptance of agent technology.
- In Parunak and Odell (2002) the UML-based representation of social structures such as groups and roles is investigated. UML conventions and AUML extensions are proposed that support the use of social structures in analysis and design.
- In Lind (2001) the specification of agent interaction protocols through standard UML is proposed.
- In Depke *et al*. (2001) an approach to the modelling of agents that uses UML and graph transformation is described.
- In Depke *et al*. (2000) a proposal for the integration of agent roles in UML can be found.
- In Bergenti and Poggi (1999) an approach to the UML-based modelling of multi-agent architectures and ontologies for agent-agent communication is introduced.

General considerations on the requirements for an ideal "unified agent-oriented modelling language" (UAML) should fulfill are provided in Arazy and Woo (2000) (Section 5).

8 Conclusions

AOSE is an important and exciting field emerging at the intersection of agent-based computing and software engineering. This field deals with practical and theoretical aspects and facets of software that possesses key characteristics of agents and multi-agent systems such as goal-directed autonomous activity and peer-to-peer interaction in cooperative and competitive settings. In particular, AOSE does not only concern a few specific developmental activities but covers all phases of software development, ranging from early requirements engineering to maintenance. Moreover, as recent developments in the field indicate, agent orientation as pursued by AOSE is in some sense twofold, concerning both the software itself and the engineering process – a duality that is well captured by the slogans "agent-oriented software (and its engineering)" and "agent-oriented engineering (of software)". The current main foci of the field are analysis and design methods, development tools and languages for programming and communication. It is important to see that AOSE does not question *general* software engineering techniques, principles and solutions (including, for example, basics such as structured

development, life-cycle models, patterns and software project management guidelines), as most of them do apply to agent-oriented software as well. This is not surprising, simply because agent-oriented software *is* software and agent-oriented engineering *is* engineering. What AOSE questions, however, is the suitability of available *specific* (e.g. object-oriented) software-technical approaches for capturing and comfortably handling agent orientation with all its characteristics and implications. AOSE aims at satisfying the need for approaches that are specifically tailored for "agent-oriented engineering" and "agent-oriented software".

Agent orientation in software engineering possesses a highly innovative scientific and technological potential and the capacity to produce novel perspectives and first-rate solutions to a broad range of complex applications. This is the main reason for the steadily growing interest in AOSE. The field has experienced rapid development and enormous progress in recent years. Despite this, and with regard to the state of the art in the field, it can be said that most available approaches – methods and frameworks, developmental tools and languages –are still in an early prototypical stage which is characterised by an emphasis on experimental and conceptual exploration and/or by a lack of systematical testing. The following lines of future practical and theoretical work are identified as being particularly important:

- Further clarification of the notion of agency and the meaning of agent-specific key concepts such as role, group and organisation. This includes the specification and refinement of these concepts in terms of software-technical requirements.
- Further clarification of the unique characteristics of agent orientation and agent-oriented software. This includes an in-depth analysis of the relationships between agent orientation and object orientation, and the specification of precise guidelines for identifying applications that demand agent-oriented solutions. Moreover, as autonomy is a main feature of agency, it also includes a careful analysis of organisational, economic, social and legal consequences of integrating agent-oriented software into decision and business processes.
- Development of industrial-strength methods, frameworks and tools for building agent-oriented software which are more concrete and detailed than currently available agent-oriented approaches.
- Further standardisation efforts with regard to agent-specific languages, interaction protocols, and specification and representational formalisms. This includes, in particular, the development of communication languages having unambiguous formal semantics.

Progress along these lines is a necessary prerequisite for a widespread use of agent technology in industrial and commercial applications in general and for the establishment of agent orientation as a significant or even dominant paradigm in software engineering in particular. This widespread technological use and this paradigmatic establishment constitute a very challenging goal. It is realistic to assume that this goal can be achieved, although not within the next few years. Both researchers and practitioners should protect themselves against unrealistic expectations towards the rate of future advancement of AOSE – and in this respect it may be useful to keep in mind that the establishment of object orientation as a mainstream paradigm in software engineering did not happen overnight but has been a scientific and commercial process that took around twenty years.

References

ABS, 1999, *Agent Building Shell* `http://www.eil.utoronto.ca/abs-page/abs-overview.html`.

AgentBuilder, 1999, `http://www.agentbuilder.com/`.

agenTool, 2000, `http://www.cis.ksu.edu/~sdeloach/ai/agentool.htm`.

Aksit, M and Bergmans, L, 1992, "Obstacles in object-oriented software development" in *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications* (OOPSLA'92) 341–358.

Arazy, O and Woo, CC, 2000, "Analysis and design of agent-oriented information systems" Working Paper 99-MIS-004, Faculty of Commerce and Business Administration, University of British Columbia. Revised version to appear in Knowledge Engineering Review.

Arbab, F, Ciancarini, P and Hankin, C, 1998, "Coordination languages for parallel programming" *Parallel Computing* **24**(7) 989–1004.

Aridor, Y and Lange, V, 1998, "Agent design patterns: elements of agent application design" in *Proceedings of the Second International Conference on Autonomous Agents* (Agents'98) 108–115.

Barbuceanu, M and Fox, MS, 1996, "Capturing and modeling coordination knowledge for multiagent systems" *International Journal of Cooperative Information Systems* **5**(2–3) 275–314.

Bauer, B, Müller, JP and Odell, J, 2001, "Agent UML: a formalism for specifying multiagent software systems" *Agent-Oriented Software Engineering: Proceedings of the First International Workshop* (AOSE-2000) 91–103.

Bauer, M, 2002, "UML class diagrams revisited in the context of agent-based systems" in MJ Wooldridge, G Weiß and P Ciancarini (eds) *Agent-oriented software engineering: Proceedings of the Second International Workshop* (AOSE-2001) (Lecture Notes in Artificial Intelligence, Volume 2222).

Benerecetti, M, Giunchiglia, F and Serafini, L, 1999, "A model checking algorithm for multi-agent systems" *Intelligent Agents V Proceedings of the Fifth International Workshop on Agent Theories, Architectures and Languages* (ATAL-98) 163–176.

Bergenti, F and Poggi, A, 1999, "A development environment for the realization of open and scalable multi-agent systems" *Multi-Agent System Engineering: Proceedings of the Ninth European Workshop on Modelling Autonomous Agents in a Multi-Agent World* (MAAMAW-99) 52–62.

Booch, G, 1994, *Object-Oriented Analysis and Design with Applications* (2nd edition) Addison Wesley.

Bordiga, A, Brachman, RJ, McGuinness, DL and Resnick, LA, 1989, "CLASSIC: a structural data model for objects" in *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data* 59–67.

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

Bradshaw, JM (ed.), 2002, *Handbook of Agent Technology* AAAI Press/The MIT Press.

Brauer, W, Nickles, M, Rovatsos, M, Weiß, G and Lorentzen, KF, 2002, "Expectation-oriented analysis and design" *Agent-Oriented Software Engineering: Proceedings of the Second International Workshop* (AOSE-2001) 226–244.

Brazier, FMT, Dunin-Keplicz, BM, Jennings, NR and Treur, J, 1997, "DESIRE: modelling multi-agent systems in a compositional framework" *International Journal of Cooperative Information Systems*, **6**(1) 67–94.

Brooks, FP, 1986, "No silver bullet" *Proceedings of the IFIP Tenth World Computer Conference* 1069–1076.

Bryson, J and McGonigle, B, 1998, "Agent architecture as object oriented design" *Intelligent Agents IV Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages* (ATAL-97) 15–30.

Buhr, R, Amyot, D, Elammari, M, Quesnel, D, Gray, T and Mankovski, S, 1998a, "High-level, multi-agent prototypes from a scenario-path notation: a feature-interaction example" in *Proceedings of the Third International Conference on the Practical Application of Intelligent Agents and Multi-Agent Systems* (PAAM'98) 277–298.

Buhr, R, Elammari, M, Gray, T and Mankovski, S, 1998b, "Applying use case maps to multi-agent systems: a feature interaction example" in *Hawaii International Conference on System Sciences* (HICSS'98) **6** 171–179.

Burkhard, HD, 1995, "Agent-oriented programming in open systems" in MJ Wooldridge and NR Jennings (eds) *Intelligent Agents* (Lecture Notes in Artificial Intelligence, Volume 890) Springer-Verlag.

Burmeister, B, 1996, "Models and methodology for agent-oriented analysis and design" in K Fischer (ed.) *Working Notes of the KI96 Workshop on Agent-oriented Programming and Distributed Systems* DFKI Dokument D-96–06.

Buschmann, F, Meunier, R, Rohnert, H, Sommerlad, P and Stal, M, 1996, *Pattern-Oriented Software Architecture: A System of Patterns*. Wiley.

Busetta, P, Rönnquist, R, Hodgson, A and Lucas, A, 1999, "JACK Intelligent Agents: components for intelligent agents in Java. *Agentlink News* **2** 2–5.

Chung, L, Nixon, BA, Yu, E and Mylopoulos, J, 2000, *Non-Functional Requirements in Software Engineering*. Kluwer Academic Press.

Ciancarini, P, 1996, "Coordination models and languages as software integrators" *ACM Computing Surveys*, **28**(2) 300–302.

Ciancarini, P and Wooldridge, M (eds), 2001, *Agent-Oriented Software Engineering: Proceedings of the First International Workshop* (AOSE-2000) (Lecture Notes in Computer Science, Volume 1957) Springer-Verlag.

Cockburn, D and Jennings, NR, 1996, "ARCHON: a distributed artificial intelligence system for industrial applications" in GMP O'Hare and NR Jennings (eds) *Foundations of Distributed Artificial Intelligence* Wiley.

Cohen, PR and Levesque, HJ, 1990, "Intention is choice with commitment" *Artificial Intelligence* **42** 213–261.

Cost, RS, Soboroff, I, Lakhani, J, Finin, T, Miller, E and Nicholas, C, 1998, "TKQML: a scripting tool for building agents" *Intelligent Agents IV Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages* (ATAL-97) 339–343.

Crabtree, B, 1998, "What chance software agents?" *Knowledge Engineering Review*, **13**(2) 131–136.

Cremonini, M, Omicini, A and Zambonelli, F, 1999, "Multi-agent systems on the Internet: extending the scope of coordination towards security and topology" *Multi-Agent System Engineering: Proceedings of the Ninth European Workshop on Modelling Autonomous Agents in a Multi-Agent World* (MAAMAW-99) 77–88.

DAML-LANGUAGE, 2001, `http://www.daml.org/language/`.

Dardenne, A, van Lamsweerde, A and Fickas, S, 1993, "Goal-directed requirements acquisition" *Science of Computer Programming* **20** 3–50.

Davies, WHE and Edwards, P, 1994, "AGENT-K: an integration of AOP and KQML" *Proceedings of the CIKM'94 Workshop on Intelligent Agents.*

De Giacomo, G, Lespérance, Y and Levesque, V, 2000, "ConGolog: a concurrent programming language based on the situation calculus" Artificial Intelligence **121** 109–169.

Decker, KS, 1996, "Distributed artificial intelligence testbeds" in GMP O'Hare and NR Jennings (eds) *Foundations of Distributed Artificial Intelligence* John Wiley & Sons.

Decker, KS, Durfee, EH and Lesser, VR, 1989, "Evaluating research in cooperative distributed problem solving" in MN Huhns and L Gasser (eds) *Distributed Artificial Intelligence* Volume 2 Pitman/Morgan Kaufmann.

Denti, E and Omicini, A, 1999, "Engineering multi-agent systems in LuCe" *Proceedings of the ICLP'99 International Workshop on Multi-Agent Systems in Logic Programming* (MAS'99).

Depke, R, Engels, G and Küster, JM, 2000, "On the integration of roles in UML" Technical Report No. 214, University of Paderborn, Germany.

Depke, R, Heckel, R and Küster, JM, 2001, "Improving the agent-oriented modelling process with roles" *Proceedings of the Fifth International Conference on Autonomous Agents* (Agents'01) 640–647.

Deugo, D, Oppacher, F, Kuester, J and Otte, IV, 1999, "Patterns as a means for intelligent software engineering" *Proceedings of the International Conference on Artificial Intelligence* (IC-AI'99) Volume II 605–611.

Deugo, D, Weiss, M and Kendall, E, 2000, "Reusable patterns for agent coordination" in A Omicini, F Zambonelli, M Klusch and R Tolksdorf (eds) *Coordination of Internet Agents: Models, Technologies and Applications* Springer-Verlag.

Dignum, F, 2000, "Agent communication and cooperative information agents" *Cooperative Information Agents IV Proceedings of the Fourth International Workshop on Cooperative Information Agents* (CIA-2000) 191–207.

Dignum, V, Weigand, H and Xu, L, 2002, "Agent societies: toward frameworks-based design" *Agent-Oriented Software Engineering: Proceedings of the Second International Workshop* (AOSE-2001) 33–49.

D'Inverno, M and Luck, M (eds), 2001, *Understanding agent systems* Springer-Verlag.

D'Inverno, M and Luck, M, 1998, "Engineering AgentSpeak(L): a formal computational model" *Journal of Logic and Computation* **8**(3) 233–260.

D'Inverno, M, Fisher, M, Lomuscio, A, Luck, M, de Rijke, M, Ryan, M and Wooldridge, M, 1997, "Formalisms for multi-agent systems" *Knowledge Engineering Review*, **12**(3) 315–321.

Dooley, RA, 1976, "Appendix B: repartee as a graph" in RE Longacre (ed.) *An Anatomy of Speech Notions* Peter de Ridder.

Drogoul, A and Collinot, A, 1998, "Applying an agent-oriented methodology to the design of artificial organizations: a case study in robotic soccer" *Autonomous Agents and Multi-Agent Systems* **1**(1) 113–129.

Du Bois, P, 1998, 1997, "The Albert II reference manual" Technical Report RR-97–002, Computer Science Department, University of Namur.

Dubois, E, Du Bois, P, Dubru, F and Petit, M, 1994, "Agent-oriented requirements engineering: a case study using the albert language" *Proceedings of the Fourth International Working Conference on Dynamic Modelling and Information Systems* (DYNMOD'94) 205–238.

Elammari, M and Lalonde, W, 1999, "An agent-oriented methodology: high-level and intermediate models" *First International Workshop on Agent-Oriented Information Systems* (AOIS'99).

Erdmann, M and Studer, R, 1998, "Use-cases and scenarios for developing knowledge-based systems" *Proceedings of the 15th IFIP World Computer Congress* (WCC'98), *Conference on Information Technologies and Knowledge Systems* 259–272.

EURESCOM/MESSAGE, July 2000, EURESCOM (European Institute for the Research and Strategic Studies in Telecommunications) Project on a Methodology for Engineering Systems of Software Agents (MESSAGE), Deliverable 1: Initial Methodology.

Feather, M, 1987, "Language support for the specification and development of composite systems" *ACM Transactions on Programming Languages and Systems* **9**(2) 198–234.

Fensel, D, 2001, *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce* Springer-Verlag.

Ferber, J and Gutknecht, O, 1998, "A meta-model for the analysis and design of organizations in multi-agent systems" *Proceedings of the 3nd International Conference on Multi-Agent Systems* (ICMAS-98) 128–135.

Ferber, J, Gutknecht, O, Jonker, CM, Müller, J-P and Treur, J, 2000, "Organization models and behavioural requirements specification for multi-agent systems" *Notes of the ECAI2000 Workshop on Modelling Artificial Societies and Hybrid Organizations* (MASHO) 10–25.

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

FIPA, 1999, FIPA (Foundation for Intelligent Agents) `http://www.fipa.org`.

FIPA-OS, 2000, `http://fipa-os.sourceforge.net/`.

Fischer G, Redmiles, D, Williams, L, Puhr, G, Aoki, A and Nakakoji, K, 1995, "Beyond object-oriented technology: where current object-oriented approaches fall short" *Human-Computer Interaction* **10**(1) 79–119.

Fisher, M, 1995, "Representing and executing agent-based systems" in MJ Wooldridge and NR Jennings (eds) *Intelligent Agents* (Lecture Notes in Artificial in Artificial Intelligence, Volume 890) Springer-Verlag.

Fisher, M, 1997, "If Z is the answer, what could the question possibly be?" In JP Müller, MJ Wooldridge and NR Jennings (eds) *Intelligent Agents III* (Lecture Notes in Artificial Intelligence, Volume 1193) Springer-Verlag.

Fisher, M and Wooldridge, M, 1997, "On the formal specification and verification of multi-agent systems" *International Journal of Cooperative Information Systems* **6**(1) 37–65.

Fisher, M, Müller, J, Schroeder, M, Staniford, G and Wagner, G, 1997, "Methodological foundations for agent-based systems" *Knowledge Engineering Review* **12**(3) 323–329.

Franklin, S and Graesser, A, 1997, "Is it an agent, or just a program?: a taxonomy for autonomous agents" in JP Müller, MJ Wooldridge and NR Jennings (eds) *Intelligent Agents III* (Lecture Notes in Artificial Intelligence, Volume 1193) Springer-Verlag.

Gadomski, A, 1993, "TOGA: a methodological and conceptual pattern for modeling abstract intelligent agents" in *Proceedings of the First AIA Round-Table on Abstract Intelligent Agents*.

Gamma, E, Helm, R, Johnson, R and Vlissides, J (eds), 1995, *Design Patterns* Addison-Wesley.

Garijo, FJ and Boman, M (eds), 1999, *Multi-Agent System Engineering: Proceedings of the Ninth European Workshop on Modelling Autonomous Agents in a Multi-Agent World* (MAAMAW-99) (Lecture Notes in Artificial Intelligence Volume 1647) Springer-Verlag.

Gelernter, D and Carriero, N, 1992, "Coordination languages and their significance" *Communications of the ACM* **35**(2) 97–107.

Genesereth, MR and Fikes, RE, 1992, "Knowledge Interchange Format. Version 3.0, Reference Manual" Technical Report Logic-92–1, Computer Science Department, Stanford University.

Genesereth, MR and Ketchpel, SP, 1994, "Software agents" *Communications of the ACM* **37**(7) 48–53 and 147.

Georgeff, MP and Lansky, AL, 1987, "Reactive reasoning and planning" *Proceedings of the 6th National Conference on Artificial Intelligence* (AAAI-87) 677–682.

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

Glaser, N, 1996, "Contribution to knowledge modelling in a multi-agent framework" Ph.D. thesis, Université Henry Poincaré.

Grasshopper, 1998, `http://www.grasshopper.de/index.html`.

Gruber, TR, 1992, "Ontolingua: a mechanism to support portable ontologies" Technical Report KSL-91–66, Knowledge Systems Laboratory, Stanford University.

Gruber, TR, 1995, "Toward principles for the design of ontologies used for knowledge sharing" *International Journal of Human-Computer Studies* **41** 399–424.

Gustavsson, R, 1994, "Agent oriented software engineering: a motivation for and an introduction to a novel approach to modeling and development of open distributed systems" Technical Report 5/94, Department of Computer Science and Business Administration, University of Karlskrona/Ronneby.

Gustavsson, V, 1998, "Multi agent systems as open societies: a design framework" *Intelligent Agents IV Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages* 329–337.

Hindriks, K v, de Boer, FS, van der Hoek, W and Meyer, J-J, 1998, "A formal embedding of AgentSpeak(L) in 3APL" in G Antoniou and J Slaney (eds) *Advanced Topics in Artificial Intelligence* (Lecture Notes in Artificial Intelligence, Volume 1502) Springer-Verlag.

Hindriks, K v, de Boer, FS, van der Hoek, W and Meyer, J-J, 1999, "Agent programming in 3APL" *Autonomous Agents and Multi-Agent Systems* **2**(4) 357–401.

Hindriks, K v, Lespérance, Y and Levesque, HJ, 2000, "A formal embedding of ConGolog in 3APL" *Proceedings of the 14th European Conference on Artificial Intelligence* (ECAI-2000) 558–562.

Holvoet, T, 1996, "Synchronization specifications for agents with net-based behavior description" *Proceedings of CESA'96 (Computational Engineering in Systems Applications), Symposium Discrete Events and Manufacturing Systems* 613–618.

Horn, E, Kupries, M and Reinke, T, 1998, "Object-oriented software architecture types for the substantiation, development and facrication of agent application systems" *Proceedings of the Eleventh International Conference on Software Engineering and its Applications*.

Huhns, MN, 2001, "Interaction-oriented programming" *Agent-Oriented Software Engineering: Proceedings of the First International Workshop* (AOSE-2000) 29–44.

IBM Aglets Development Kit, 2000, http://www.trl.ibm.com/aglets/.

Iglesias C, Garijo, M, Gonzales, JC and Velasco, JR, 1998, "Analysis and design of multi-agent systems using MAS-CommonKADS" *Intelligent Agents IV Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages* (ATAL-97) 313–326.

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

Intelligent Agent Factory, 2000, http://www.bitpix.com.

JACK, 1998, *JACK Intelligent Agents*, http://www.agent-software.com.au/shared/home/index.html.

Jacobson, I, Christerson, M, Jonsson, P and Övergaard, G, 1994, *Object-Oriented Software Engineering: A Use Case Driven Approach* Addison-Wesley (Revised Printing).

JADE, 1999, http://sharon.cselt.it/projects/jade/.

JAFMAS, 2000, http://www.ececs.uc.edu/~abaker/jafmas/.

JATLite, 2000, http://java.stanford.edu/java_agent/html/.

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

Jennings, NR and Wooldridge, M, 2002, "Agent-oriented software engineering" in J Bradshaw (ed.) *Handbook of Agent Technology*. AAAI/MIT Press.

Kendall, EA, 1998, "Agent roles and role models: new abstractions for multiagent system analysis and design" *International Workshop on Intelligent Agents in Information and Process Management*.

Kendall, EA and Malkoun, MT, 1996, "The layered agent patterns" in *Pattern Languages of Programs* (PLoP'96).

Kendall, EA, Malkoun, MT and Jiang, C, 1997a, The application of object-oriented analysis to agent based systems. "The Report on Object Oriented Analysis and Design in Conjunction with The Journal of Object-Oriented Programming".

Kendall, EA, Pathak, CV, Krishna, PVM and Suresh, CB, 1997b, "The layered agent pattern language" in *Proceedings of the Conference on Pattern Languages of Programs* (PLoP'97).

KIF, 1999, http://www.cs.umbc.edu/kse/kif/.

Kifer, M, Lausen, G and Wu, J, 1995, "Logical foundations of object-oriented and frame-based languages" *Journal of the ACM* **42**(4) 741–843.

Kinny, D, 1999, "The Agentis agent interaction model" *Intelligent Agents V Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages* (ATAL-98) 331–344.

Kinny, D, Georgeff, M and Rao, A, 1996, "A methodology and modelling technique for systems of BDI agents" *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World* (MAAMAW-96) 56–71.

Kolp, M, Castro, J and Mylopoulos, J, 2001a, "A social organization perspective on software architectures" *Proceedings of the First International Workshop From Software Requirements to Architectures* (STRAW 01) at ICSE 2001.

Kolp, M, Giorgini, P and Mylopoulos, J, 2001b, "An goal-based organizational perspective on multi-agents architectures" *Proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages* (ATAL-2001).

Kone, MT, Shimazu, A and Nakajima, T, 2000, "The state of the art in agent communication languages" *Knowledge and Information Systems*, **2** 259–284.

KQML, 1999, *The UMBC KQML Web* http://www.cs.umbc.edu/kqml/.

Kuwabara, D, Ishida, T and Osato, N, 1995, "AgentTalk: coordination protocol description for multiagent systems" *Proceedings of the First International Conference on Multi-Agent Systems* (ICMAS-95) 455.

Larman, C, 1997, *Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design*. Prentice Hall, Englewood Cliffs, NJ.

LEAP, 2000, http://leap.crm-paris.com/.

Lenat, DB and Guha, RV (eds), 1990, *Building Large Knowledge-Based Systems. Representation and Inference in the Cyc Project*. Addison-Wesley.

Lind, J, 2001a, *Iterative Software Engineering for Multiagent Systems: The MASSIVE Method* (Lecture Notes in Computer Science, Volume 1994) Springer-Verlag.

Lind, J, 2001b, "Specifying agent interaction protocols with standard UML" *Agent-Oriented Software Engineering: Proceedings of the Second International Workshop* (AOSE-2001) 136–147.

Luck, M, 1999, "From definition to deployment: What next for agent-based systems?" *Knowledge Engineering Review*, **2** 119–124.

McCabe, FG and Clark, KL, 1995, "April – Agent PRocess Interaction Language" in MJ Wooldridge and NR Jennings (eds) *Intelligent Agents* (Lecture Notes in Artificial in Artificial Intelligence, Volume 890) Springer-Verlag.

MacGregor, R, 1994, "A description classifier for the predicate calculus" *Proceedings of the Twelfth National Conference on Artificial Intelligence* (AAAI-94) 213–220.

MADKIT, 1999, *Multi-Agent Development KIT*, http://www.madkit.org/.

Manna, Z and Pnueli, A (eds), 1995, *Temporal Verification of Reactive Systems* Springer-Verlag.

Martin, DL, Cheyer, AJ and Moran, DB, 1999, "The open agent architecture: a framework for building distributed software systems" *Applied Artificial Intelligence* **13**(1–2) 91–128.

Microsoft Agent, 2000, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msagent/agentstartpage_7=gdh.asp.

Moldt, D and Wienberg, F, 1997, "Multi-agent systems based on coloured Petri nets" *Proceedings of the 18th International Conference on Application and Theory of Petri Nets* (ICATPN'97) 82–101.

Moss, S, Gaylard, H, Wallis, S and Edmonds, B, 1996, "SDML: a multi-agent language for organizational modelling" CPM Report 97–19, Centre for Policy Modelling, Manchester Metropolitan University.

Moulin, B and Brassad, M, 1996, "A scenario-based design method and an environment for the development of multiagent systems" *Proceedings of the First Australian Workshop on DAI* 216–296.

Mulder, M, Treur, J and Fisher, M, 1998, "Agent modelling in Metatem and DESIRE" *Intelligent Agents IV Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages* (ATAL-97) 193–208.

Müller, J, 1996, "Towards agent systems engineering" *International Journal on Data and Knowledge Engineering* **23** 217–245.

Müller, JP, 1996, "Control architectures for autonomous and interacting agents: a survey" in L Cavedon, L Rao and W Wobcke (eds) *Intelligent Agents Systems: Theoretical and Practical Issues* (Lecture Notes in Artificial in Artificial Intelligence, Volume 1209) Springer-Verlag.

Müller, JP, 1999, "The right agent (architecture) to do the right thing" in JP Müller, MP Singh and AS Rao (eds) *Intelligent Agents V* (Lecture Notes in Artificial in Artificial Intelligence, Volume 1555) Springer-Verlag.

Mylopoulos, J and Castro, J, 2000, "Tropos: a framework for requirements-driven software development" in J Brinkkemper and A Solvberg (eds) *Information Systems Engineering: State of the Art and Research Themes* Springer-Verlag.

Mylopoulos, J, Chung, L and Yu, ESK, 1999, "From object-oriented to goal-oriented requirements analysis" *Communications of the ACM* **42**(1) 31–37.

Ndumu, DT, Nwana, HS, Lee, L and Collins, J, 1999, "Visualising and debugging distributed multi-agent systems" *Proceedings of the Third International Conference on Autonomous Agents* (Agents'99) 326–333.

Nuseibeh, BA and Easterbrook, SM, 2000, "Requirements engineering: a roadmap" *Proceedings of the 22nd International Conference on Software Engineering* (ICSE'00) 35–46.

Nwana, HS, 1996, "Software agents: an overview" *Knowledge Engineering Review* **11**(3) 205–244.

O'Hare, GMP, 1996, "Agent factory: an environment for the fabrication of multiagent systems" in GMP O'Hare and NR Jennings (eds) *Foundations of Distributed Artificial Intelligence* Wiley.

O'Hare, GMP and Wooldridge, M, 1992, "A software engineering perspective on multi-agent system design" in N Avouris and L Gasser (eds) *Distributed Artificial Intelligence: Theory and Practice* Kluwer Academic Publishers.

OAA, 1999, *Open Agent Architecture*, http://www.ai.sri.com/~oaa/.

Odell, J, September 1999, *Objects and agents: How do they differ?, Working Paper v2.2* (http://www.jamesodell.com).

Odell, J, Parunak, V and Bauer, B, 2001, "Representing agent interaction protocols in UML" *Agent-Oriented Software Engineering: Proceedings of the First International Workshop* (AOSE-2000) 121–140.

OIL, 2001, http://www.ontoknowledge.org/oil/.

OMG MASIF Standard, 1999, http://www.fokus.gmd.de/research/cc/ecco/masif/.

Omicini, A, 2000, "On the semantics of tuple-based coordination models" *Proceedings of the ACM Symposium on Applied Computing* (SAC'99) 175–182.

Omicini, A, 2001, SODA: societies and infrastructures in the analysis and design of agent-based systems" *Agent-Oriented Software Engineering.Proceedings of the First International Workshop* (AOSE-2000) 185–194.

Omicini, A and Zambonelli, F, 1999, "Coordination for Internet application development" *Autonomous Agents and Multi-Agent Systems* **2**(3) 251–269.

Omicini, A, Tolksdorf, R and Zambonelli, F (eds), 2000a, *Engineering Societies in the Agents World: Proceedings of the First International Workshop* (ESAW 2000) (Lecture Notes in Artificial Intelligence, Volume 1972) Springer-Verlag.

Omicini, A, Zambonelli, F, Klusch, M and Tolksdorf, R (eds), 2000b, *Coordination of Internet Agents: Models, Technologies, Applications* Springer-Verlag.

ONTOEDIT, 2001, http://www.ontoprise.com.

Papadopoulos, GA and Arbab, F, 1998, "Coordination models and languages" *Advances in Computers* **46** 329–400.

Parunak, V, 1996, "Visualizing agent conversations: using enhanced Dooley graphs for agent design and analysis" *Proceedings of the 2nd International Conference on Multi-Agent Systems* (ICMAS-96) 275–282.

Parunak, V, 1999, "Industrial and practical applications of DAI" in G Weiss (ed.) *Multiagent Systems* The MIT Press.

Parunak, V, 2000, "Agents in overalls: experiences and issues in the development and deployment of industrial agent-based systems" *International Journal of Cooperative Information Systems* **9**(3) 209–227.

Parunak, V and Odell, J, 2002, "Representing social structures in UML" *Agent-Oriented Software Engineering: Proceedings of the Second International Workshop* (AOSE-2001) 1–16.

Petrie, C, 2001, "Agent-based software engineering" *Agent-oriented software engineering: Proceedings of the First International Workshop* (AOSE-2000) 59–76.

Pitt, J and Mamdani, A, 1999, "A protocol-based semantics for an agent communication language" *Proceedings of the 16th International Joint Conference on Artificial Intelligence* (IJCAI-99).

Pnueli, A, 1986, "Specification and development of reactive systems" in H.-J. Kugler (ed.) *Information Processing* **86** 845–858 Elsevier Science Publ.

Poggi, A, 1995, "Daisy: an object-oriented system for distributed artificial intelligence" in MJ Wooldridge and NR Jennings (eds) *Intelligent Agents* (Lecture Notes in Artificial Intelligence, Volume 890) 341–354 Springer-Verlag.

Pont, M and Moreale, E, 1996, "Towards a practical methodology for agent-oriented software engineering with C++ and Java" Technical Report 96–33, Department of Engineering, Leicester University.

PROTEGE, 2001, `http://protege.stanford.edu/index.shtml`.

Rao, AS, 1996, "AgentSpeak(L): BDI agents speak out in a logical computable language" *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World* (MAAMAW-96) 42–55.

Rao, AS and Georgeff, MP, 1993, "A model-theoretic approach to the verification of situated reasoning systems" *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (IJCAI-93).

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

Reinke, T, 2000, "Architecture-based construction of multiagent systems" *Notes of the ECAI2000 Workshop on odelling Artificial Societies and Hybrid Organizations* (MASHO-2000) 99–111.

RETSINA, 2000, `http://www-2.cs.cmu.edu/~softagents/`.

Ricordel, P-M and Demazeau, Y, 2000, "From analysis to deployment: a multi-agent platform survey" *Working Notes of the First International Workshop on Engineering Societies in the Agents' World* (ESAW-00) 93–105.

Roscoe, AW (ed.), 1997, *Theory and practice of concurrency* Prentice Hall.

Rosenschein, SJ and Kaelbling, LP, 1986, "The synthesis of digital machines with provable epistemic properties" *Proceedings of the Conference on Theoretical Aspects of Reasoning About Knowledge* 83–98.

Russell, SJ and Norvig, P, 1995, *Artificial Intelligence. A Modern Approach*. Prentice Hall.

Sadek, MD, 1991, "Dialogue acts are rational plans" *Proceedings of the ESCA/ETRW Workshop on the Structure of multimodal Dialogue* 1–29.

Sadek, MD, Bretier, P and Panaget, F, 1997, "ARTIMIS: natural dialogue meets rational agency" *Proceedings of the 15th International Joint Conference on Artificial Intelligence* (IJCAI-97) 1030–1035.

Satapathy, G and Kumara, SRT, 1999, "Object oriented design based agent modeling" *Proceedings of the Fourth International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology* (PAAM'99) 143–162.

Schild, K, 1999, "On the relationship between BDI logics and standard logics of concurrency" *Intelligent Agents V Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages* (ATAL-98) 47–61.

Schreiber, G, Akkermans, H, Anjewierden, A, de Hoog, R, Shadbolt, N, van de Velde, W and Wielinga, B, 1999, *Knowledge Engineering and Management. The CommonKADS Methodology* The MIT Press.

Schumacher, M, Chantemargue, F and Hirsbrunner, B, 1999, "The STL++coordination language: a base for implementing distributed multi-agent applications" *Proceedings of the Third International Conference on Coordination Languages and Models* (COORDINATION'99) 399–414.

Shaw, M and Garlan, D, 1996, "*Software Architecture: Perspectives on an Emerging Discipline* Prentice Hall.

SHOE, 2001, `http://www.cs.umd.edu/projects/plus/shoe/`.

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

Silva, A and Delgado, J, 1998, "The agent pattern: a design pattern for dynamic and distributed applications" *Proceedings of the European Conference on Pattern Languages of Programming and Computing* (EuroPLoP'98).

SIM_AGENT, 1996, `http://www.cs.bham.ac.uk/~axs/cog_affect/sim_agent.html`.

Singh, MP, 1996, "Toward interaction-oriented programming" *Proceedings of the 2nd International Conference on Multi-Agent Systems* (ICMAS-96) 457.

Singh, MP, 1998, "Agent communication languages: rethinking the principles" *IEEE Computer* **31**(12) 55–61.

Singh, MP, 2000, "Synthesizing coordination requirements for heterogeneous autonomous agents" *Autonomous Agents and Multi-Agent Systems* **3**(2) 107–132.

Singh, MP, Huhns, MN and Stephens, LM, 1993, "Declarative representations of multiagent systems" *IEEE Transactions on Knowledge and Data Engineering* **5**(5) 721–739.

Spivey, JM, 1992, *The Z Notation* Prentice Hall (2nd edition).

Steiner, DD, 1996, "IMAGINE: an integrated environment for constructing distributed artificial intelligence systems" in GMP O'Hare and NR Jennings (eds) *Foundations of Distributed Artificial Intelligence* 345–364 Wiley.

Studer, R, Benjamins, V and Fensel, D, 1998, "Knowledge engineering: principles and methods" *IEEE Transactions on Data and Knowledge Engineering* **25** 161–197.

Subrahmanian, VS, Bonatti, P, Dix, J, Eiter, T, Kraus, S, Ozcan, F and Ross, R, 2000, *Heterogeneous agent systems* The MIT Press.

Swarm, 2000, *Swarm Development Group*, `http://www.swarm.org/`.

Tahara, Y, Ohsuga, A and Honiden, S, 1999, "Agent system development method based on agent patterns" *Proceedings of the International Conference on Software Engineering* 356–367.

Thomas, SR, 1995, "The PLACA agent programming language" in MJ Wooldridge and NR Jennings (eds) *Intelligent Agents* (Lecture Notes in Artificial in Artificial Intelligence, Volume 890) Springer-Verlag.

Uschold, M and Gruninger, M, 1996, "Ontologies: principles, methods and applications" *Knowledge Engineering Review* **11**(2) 93–155.

Van Eck, P, Engelfriet, J, Fensel, D, van Harmelen, F, Venema, Y and Willems, M, 2001, "A survey of languages for specifying dynamics: a knowledge engineering perspective" *IEEE Transactions on Knowledge and Data Engineering* **13**(3) 462–496.

Van Lamsweerde, A, 2000, "Requirements engineering in the year 00: a research perspective" *Proceedings of the 22nd International Conference on Software Engineering* (ICSE'00) 5–19.

Verharen, EM, Dignum, F and Bos, S, 1998, "Implementation of a cooperative agent architecture based on the language-action perspective" *Intelligent Agents IV Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages* (ATAL-97) 31–44.

Wagner, G, 1996, "VIVA knowledge-based agent programming" Preprint, Institut für Informatik, Universität Leipzig.

Wagner, G, 2000, "The Agent-Object-Relationship meta-model: towards a unified conceptual view of state and dynamics" Technical report, Faculty of Technology Management, Eindhoven University of Technology.

WEBONTO, 2001, `http://webonto.open.ac.uk`.

Weidenhaupt, K, Pohl, M, Jarke, M and Haumer, P, 1998, "Scenario usage in system development: a report on current practice" *IEEE Software* **15**(March) 34–45.

Weiß, G (ed.), 1999, *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence* The MIT Press.

Weiß, G, 2001, "Agentenorientiertes Software Engineering" *Informatik Spektrum* **24**(2) 98–101.

Wellman, MP, 1996, "Market-oriented programming: some early lessons" in SH Clearwater (ed.) *Market-Based Control* World Scientific.

White, JE, 1997, "Mobile agents" in JM Bradshaw (ed.) *Software Agents* AAAI Press/The MIT Press.

Wood, M and DeLoach, SA, 2001, "An overview of the multiagent systems engineering methodology" *Agent-Oriented Software Engineering: Proceedings of the First International Workshop* (AOSE-2000) 207–222.

Wooldridge, MJ, 1992, "The logical modelling of computational multi-agent systems" Ph.D. thesis, Department of Computation, UMIST.

Wooldridge, MJ, 1998a, "Verifiable semantics for agent communication languages" *Proceedings of the 3nd International Conference on Multi-Agent Systems* (ICMAS-98) 349–356.

Wooldridge, MJ, 1998b, "Agents and software engineering" *AI*IA Notizie* **XI**(3) 31–37.

Wooldridge, MJ, 1999, "Intelligent agents" in G Weiss (ed.) *Multiagent Systems* The MIT Press.

Wooldridge, MJ (ed.), 2000, *Reasoning About Rational Agents* The MIT Press.

Wooldridge, MJ and Ciancarini, P, 2001, "Agent-oriented software engineering: the state of the art" *Agent-Oriented Software Engineering: Proceedings of the First International Workshop* (AOSE-2000) 1–28.

Wooldridge, MJ and Jennings, NR, 1995a, "Agent theories, architectures and languages: a survey" in MJ Wooldridge and NR Jennings (eds) *Intelligent Agents* (Lecture Notes in Artificial in Artificial Intelligence, Volume 890) Springer-Verlag.

Wooldridge, MJ and Jennings, NR, 1995b, "Intelligent agents: theory and practice" *Knowledge Engineering Review* **10**(2) 115–152.

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

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

Wooldridge, M, Weiß, G and Ciancarini, P (eds), 2002 *Agent-Oriented Software Engineering II Proceedings of the Second International Workshop* (AOSE-2001) (Lecture Notes in Computer Science, Volume 2222) Springer-Verlag.

XOL, 2001, `http://www.ontologos.org/ontology/xol.htm`.

Yu, ESK, 1997a, "Towards modelling and reasoning support for early-phase requirements engineering" *Proceedings of 3rd IEEE International Symposium on Requirements Engineering* (RE'97) 226–235.

Yu, ESK, 1997b, "Why agent-oriented requirements engineering?" *Proceedings of 3rd International Workshop on Requirements Engineering: Foundations for Software Quality.*

Yu, ESK, 2001, "Agent orientation as a modelling paradigm" *Wirtschaftsinformatik* **43**(2) 123–132.

Yu, ESK, 2002, "Agent-oriented modelling: software versus the world" *Agent-Oriented Software Engineering: Proceedings of the Second International Workshop* (AOSE-2001) 206–225.

Yu, ESK and Mylopoulos, J, 1998, "Why goal-oriented requirements engineering?" *Proceedings of the 4th International Workshop on Requirements Engineering* 15–22.

Yu, ESK, 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.

Zambonelli, F, Jennings, NR and Wooldridge, M, 2001, "Organisational abstractions for the analysis and design of multi-agent systems" *Agent-Oriented Software Engineering: Proceedings of the First International Workshop* (AOSE-2000) 235–252.

Zambonelli, F, Jennings, NR, Omicini, A and Wooldridge, M, 2000, "Agent-oriented software engineering for Internet applications" in A Omicini, F Zambonelli, M Klusch and R Tolksdorf (eds) *Coordination of Internet Agents: Models, Technologies and Applications* Springer-Verlag.

ZEUS, 1999, `http://www.labs.bt.com/projects/agents/zeus/index.htm`.