Agent Orientation in Software Engineering

Gerhard Weiß
Institut für Informatik, Technische Universität München
D-80290 München, Germany, weissg@in.tum.de

Revised Version for Knowledge Engineering Review

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 co-existed 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.

Table of Contents

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 multiagent 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 being 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. And third, AOSE is a field that evolves very rapidly. For that reason this guide can not 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 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 approaches 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 multiagent 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 languages 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 Modeling Language (UML). Finally, Section 8 summarizes key

[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 realizing one or several phases of the software life cycle.

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 [1, 2], that treat various key aspects of AOSE and of the paradigma of agent-based computing in general. A recent useful overview of the state of the art in AOSE is [3]. Earlier papers offering useful initial considerations on AOSE are [4, 5, 6]. There are two collections of papers on AOSE [7, 8], and related collections with a somewhat broader engineering perspective on agent systems are [9, 10]. These collections are a good starting point for exploring the field with its various facets.

2.2 Agents and Objects

Dealing with AOSE requires to deal with the notion of agency. Many different perspectives of agency have been described and discussed, and there is nothing like an "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 [11, 12]. Examples of other readable introductory texts on software agents are [13, 14]. Well written course-level texts on computational agency are [15, Chapter 2] and [16]. Books that broadly cover agent and multiagent technology are [17, 18]. Despite dissension in detail, however, there is an increasingly broad agreement on the usefulness of characterizations of the following kind, adapted from [19]:

> 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 characterization, 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 multiagent system are used synonymously.

There is an ongoing debate on the relationships between agents and objects. Valuable papers throwing light on this issue are [1, 20]. 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.

[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 behavior 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 w.r.t. their behavioral freedom and flexibility. As stated in [3, 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 can not 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 characterizations of both "agents" and "objects" are of quite general flavor, which often makes the identification of real-world representatives of both concepts difficult.

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 to take an agent-oriented approach to software is addressed in various articles. Among them are, in particular, [1, 2]. 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 characterization of the benefit of agent orientation is also emphasized in [21] 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),
- can not all be assumed to be fully controllable in their behavior (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 telecommunication 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.

As discussed by Jennings in [1], 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 [1] it is also pointed out that agent orientation fully supports the three techniques which Booch [22, pp. 16f] 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 [23]. This does hold for object orientation (see, e.g., [24, 25]), 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 realization. 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 [1], 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 organizational structures. Of course, applying such protocols and structures tends to limit agent autonomy and the potential advantages implied by it (e.g., self-organization 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 [26]. Examples of the 24 identified pitfalls are the following:

- You oversell agents.
- You see agents everywhere.
- You get religious or dogmatic about agents.
- You don't know what your agents are good for.
- You confuse buzzwords with concepts.
- You don't 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 [27, 28, 29, 30].

Work Identifying Questions that Matter. Two useful papers that highlight basic questions one has to deal with when developing industrial agent-based systems are [31, 32]. 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 [33]. Helpful guidelines for answering this question can be found in [34].

A comprehensive list of general research questions on various aspects of multiagent systems is given in [35]. 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 analyzing the functional and non-functional capabilities a software system must possess in response to real-world constraints. As noted in [36], 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 has been realized in field of requirements engineering more than a decade ago. The seminal paper in this respect is [37], in which a simple framework for modelling agents and for reasoning about their behavioral choice and constraints was introduced. Today the modelling of agents is considered as a particular area of concern in requirements engineering [38], 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* [39]. Two readable papers that clarify the role of the concept of agents in requirements engineering are [40, 41]. 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-a-modelling-construct may eventually not be materialized as an agent-as-a-software-artificat. 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**** [42]. ***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 [43] and ALBERTII [44] (ALBERT stands for "Agent-oriented Language for Building and Eliciting Real-Time requirements"). These are formal, agent-centered requirements specification languages.

A good example of how ***i**** and ALBERTII can be combined is described in [45]. A complete requirements-driven framework for agent-oriented systems development that adopts ***i**** is

- Tropos [46] (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 [47, 48]). 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 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") [49] and

- NFR ("Non-Functional Requirements") [50].

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 – i.e., 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 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 do 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 characterize 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 multiagent technology.* Approaches having this background are characterized by a clear focus on capturing social-level abstractions such as agent, group, or organization, that is, on abstractions that are above the conventional object level. These approaches are treated in 4.1.

- *Object orientation.* Approaches with this background are characterized by the attempt to appropriately extend existing object-oriented techniques such that they also capture the notion of agency. These approaches are considered in 4.2.

- *Knowledge engineering.* Approaches with this background are characterized 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 4.3.

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

4.1 Approaches based on Agent and Multiagent Technology

There are a number of "pure" methods and frameworks having agent and multiagent 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 to capture elementary characteristics of agency. Among the most often cited representatives of this type of approaches are the following:

- Gaia ("Generic Architecture for Information Availability") [51]. This is a method that distinguishes between analysis and design and associates different models with these two phases. Gaia focuses on organizational aspects in terms of concepts such as roles, interactions, and acquaintances. See also [52] which discusses an extension of Gaia based on the concept of coordination models known from the area of standard coordination languages (see 5.2).

- SODA ("Societies in Open and Distributed Agent spaces") [53]. This is another good example of an analysis and design method that concentrates on the social (inter-agent) aspects of agent systems and that employs the concept of coordination models.

- Cassiopeia [54]. This is a design method that distinguishes three levels of behavior – elementary, relational, and organizational – and aims at capturing both structural and dynamic aspects of the target system.

- AALAADIN [55]. This is a general analysis and design framework that has its focus on the organizational level of multiagent systems and is built on the three core concepts of agents, groups, and roles. A formal extension of this approach toward specifying the requirements on the overall multiagent system dynamics is proposed in [56].

Apart from these most well 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”) [57]. This is an analysis and design method that emphasizes the aspect of autonomy and that introduces expectations held by individual agents as a first-order abstraction.
- In [58] an analysis and design method is described that emphasizes the distinction between markets, networks, and hierarchies as agent society frameworks.
- In [59] a method is proposed that distinguishes between two phases – discovery and definition – within which several models of the agents’ external and internal behavior are generated. The method deals with both the visualization and specification of behavior and employs use case maps.
- In [60] is is argued that organizational abstractions – organizational rules, organizational structures, and organizational patterns – should play a central role in the analysis and design of multiagent systems. Initial considerations on an organization-oriented methodology based on these abstractions are provided.

Another elaborated agent-oriented method is Tropos (see 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., [22]) as their starting point. Prototypical examples of such approaches are the following:

- KGR [61]. This is a design and specification method for a particular class of agents, namely, BDI agents [62]. KGR extends the object modelling technique (OMT) and considers two viewpoints – external and internal – of agent systems.
- MaSE (“Multiagent Systems Engineering”) [63]. This method covers design and initial implementation through two languages called AgML (“Agent Modeling Language”) and AgDL (Agent Definition Language) and builds upon OMT and UML.
- MASSIVE (“MultiAgent SystemS Iterative View Engineering”) [64]. 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”) [65]. 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”) [66]. MASB is an analysis and design method that covers issues of both objects and agents via behavior 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 [67] 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 modeling/diagrams") [68]. 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 [69, 70, 71, 72].

4.3 Approaches based on Knowledge Engineering Technology

Knowledge engineering (e.g., [73]) 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") [74]. This is an elaborated extension of the CommonKADS methodology [75], supporting analysis, design, and automated code generation. CoMoMAS focuses on knowledge engineering issues arising in multiagent contexts and integrates a commercial tool called KADSTOOL for the conception of expertise models.

- MAS-CommonKADS ("Multi-Agent System CommonKADS") [76]. 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 [77]. By its very nature knowledge engineering has a very close relationship to the engineering of agent and multiagent 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 [78]. 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 [79];

- the belief-desire-intention model [80]; and

- $\mathcal{LORA}$ ("Logic of Rational Agents") [81].

Another logic-based specification scheme, allowing for a declarative representation of multiagent systems, is described in [82]. A well known alternative formal specification approach is

- DESIRE ("DEsign and Specification of Interacting REasoning components") [83].

This approach has its roots in knowledge engineering, although its specifications and their semantics can be formalized via temporal logics. DESIRE differs from other formal knowledge engineering specification approaches (see [84] 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 [85].

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

- axiomatic approaches, that is, approaches based on techniques of theorem proving (see [86, 87, 88] for good examples); and
- semantic approaches based on model checking (see [89, 90]).

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* [91] and
- Petri net theory.

A good example of a *Z*-based formal framework for agent system specification is described in [92]. More general considerations on the value of *Z* for agent-based system specification are provided in [93]. Good examples of Petri net-based specifications of multiagent systems can be found in [94, 95]. There is other Work in mainstream computer science that appears to be of relevance for multiagent system specification and verification. This includes, for instance, reactive systems theory [96, 97] and concurrency theory (CCS, CSP, π-calculus, etc., see [98]). The full value of these theories in multiagent 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 [63]) use formal or semi-formal standard notations to describe and to represent coordination and communication among agents, while others (e.g., Gaia [51]) 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 [99]. For instance, these graphs have been used for the explication of relationships within agent-agent conversations [100] and for the engeering of multiagent coordination requirements [101].

The reader interested in a broader discussion of the use of formalisms for multiagent systems is pointed to [102].

4.5 Further Key Issues

Hybrid Approaches. The three types of methods and frameworks that can be distinguished by applying the disciplinary background as the characterizing 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 multiagent technology may also cover critical requirements engineering issues. Hence, rather than thinking of sharp borderlines between these types, one should better 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") [103].

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 [104].

Alternative Characterizations. The disciplinary background is, of course, not the only (though a very useful) criterion for characterizing available methods and frameworks for agent-oriented software development. Five examples of alternative criteria are sketched below. None of them should be considered as "the best" because each reveals relevant properties from a different perspective. At the moment no unified and generally accepted characterization 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 [53]), although there are approaches that cover implementation as well (e.g., MASSIVE [64]).

Another alternative criterion is the *modelling level* on which a method primarily focuses. Based on this criterion, one can distinguish between approaches emphasizing 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. organizational 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 multiagent technology typlically emphasize 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 [65]) and top-down (e.g., Aalaadin [55]) 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 [54]) or BDI architectures (e.g., KGR [61]), while other approaches are more general and independent of specific agent views (e.g., Gaia [51]). 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 w.r.t. the different modelling levels mentioned above and/or w.r.t. 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 [105] compares a number of development approaches, as well as several programming languages and simulation environments, w.r.t. their coverage of life cycle phases and their abstraction granularity.

- The survey offered in [106] 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 (5.1), languages for communication and coordination among agents (5.2), and languages for specifying ontologies (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 realize 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* [107] and, more recently, *market-oriented programming* [108] and *interaction-oriented programming* [109, 110]. 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, and 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 [107] realizes the basic ideas of the agent-oriented programming paradigm as formulated by Shoham. A language that extends AGENT-0 toward planning is PLACA [111], and a language that aims at integrating AGENT-0 and KQML (see 5.2) is AGENT-K [112].
- Concurrent MetateM [113] allows to specify the intended behavior of an agent based on temporal logics. A comparison of Concurrent MetateM and DESIRE (see 4.4) is presented in [114].
- AgentSpeak(L) [115] is a rule-based language that has a 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 [116]. A formal spezification of AgentSpeak(L) based on *Z* is presented in [117].
- 3APL [118] incorporates features from imperative and logic programming. 3APL has a well defined operational semantics and supports monitoring and revising of agent goals. Work relating 3APL and AgentSpeak(L) is described in [119].
- ConGolog [120] is a concurrent logic-based language initially designed for high-level robot programming. Work relating ConGolog and 3APL is presented in [121].

Other examples of languages following the agent-oriented programming paradigm are April ("Agent PRocess Interaction Language") [122], MAIL/MAI^{2}L ("Multiagent Interaction and Implementation Language") [123], and VIVA [124]. 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 [125]. The first commercially available language for the network-independent implementation of mobile agents is Telescript [126]. A discussion of design choices for agent-oriented languages and their effects on programming open systems can be found in [127].

5.2 Languages for Communication and Coordination

The difficulty to precisely handle coordination and communication increases with the size 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") [128, 129] is perhaps the most widely used agent communication language. An integration of KQML into Tcl/Tk is proposed in [130].

- ARCOL ("ARTIMIS COmmunication Language") [131] is the communication language used in the ARTIMIS system [132]. ARCOL has a smaller set of communication primitives than KQML, but these can be composed.

- FIPA-ACL ("FIPA Agent Communication Language") [133] 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") [134, 135]. 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") [136]. COOL aims at explicitly representing and applying coordination knowledge for multiagent 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") [137] is a language that encompasses both agent-agent and agent-human communication and deals with conversational protocols and content descriptions.

- AgentTalk [138] is a coordination protocol description language for multiagent systems. AgentTalk supports the application-specific, incremental definition and customization of coordination protocols.

- CoLa ("Communication and coordination Language") [139] allows for the specification of obligations and authorizations and supports the separation of tasks and contracts.

- TuCSoN ("Tuple Centres Spread over Networks") [140] 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 [141].

- LuCe [142] 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 [143].

- STL++ ("Simple Thread Language ++") [144] is a language that provides a framework for describing the organizational structure of a multigent system. STL++ supports peer-to-peer, multicast and generative communication.

- SDML ("Strictly Declarative Modelling Language") [145] is a language designed to facilitate modelling of multiagent 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 multiagent systems. This problem is addressed in a number of publications; good examples are the following:

- in [146] it is proposed to emphasize 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 [147]; and
- the problem of determining conformance to the semantics by an independent observer is formally investigated in [148].

A brief overview and discussion of main topics of interest in agent communication research can be found in [149].

Here are pointers to some survey articles:

- A recent useful survey of intentional agent communication languages can be found in [150].
- There are several readable introductions to and reviews of coordination/communication languages and models for parallel and distributed programming; see [151, 152, 153, 154]. 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 [155].

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 edition of ontologies. Among the most elaborated examples of such languages are the following:

- Frame-based languages such as Ontolingua [156] and Frame Logic [157]. Both Ontolingua and Frame Logic extend first-order predicate logics. The key modelling primitive of these languages are frames as known from artificial intelligence.
- Description logics such as CLASSIC [158] and LOOM [159] that allow an intensional definition of concepts.
- CycL [160] 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 applications 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") [161] 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") [162] is an XML- and frame-based language for the exchange of ontologies.

- OIL ("Ontology Inference Layer") [163] 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 [164]. DAML-OIL, which replaces DAML-ONT and represents the state of the art in the field, has a well defined model-theoretic and axiomatic semantics.

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

- Protégé [165] which supports single-user ontology acquisition,

- Webonto [166] which supports multiple-user ontology acquisition over the Web, and

- OntoEdit [167] which supports multilingual development of ontologies and multiple inheritance.

A very useful and broader introduction to ontologies is [168]. Readers interested in good introductory articles on ontologies are pointed to [169, 170].

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 [171] is a toolkit that has been developed at the British Telecom Intelligent System Research Lab. A visualization tool for agent applications built with ZEUS (or other toolkits) is described in [172].

- JADE ("Java Agent DEvelopment Framework") [173] has been developed at the University of Parma, Italy.

- LEAP ("Lightweight Extensible Agent Platform") [174] is intended to be executable on small devises such as PDAs or phones. LEAP is being developed within European's Fifth Framework program by several industrial and academic contract partners (MOTOROLA, ADAC, BROADCOM, BT, Siemens, and the University of Parma).

- agenTool [175] is a Java-based graphical development environment that supports the MaSE method (see 4.2). agenTool was originally developed at the Artificial Intelligence Lab of the Air Force Institute of Technology, Ohio.

[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/

- RETSINA [176] 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") [177], which has been developed at the Stanford Center for Design, is a package of Java programs that allows to create software agents that communicate over the Internet.
- FIPA-OS [178] 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 [179] is a platform which is being developed at LIRMM (France). MADKIT is based on the Aalaadin model (see 4.1).

Other examples are SIM_AGENT [180], JAFMAS ("Java-based Agent Framework for Multi-Agent Systems") [181], ABS ("Agent Building Shell") [182] which employs the language COOL (see 5.2), OAA ("Open Agent Architecture") [183], and Agentis [184] which is a modelling framework for BDI agents.

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

- AgentBuilder [185] is a tool offered by Reticular Systems Inc., USA. AgentBuilder is available in two versions: AgentBuilder Lite (entry-level) and AgentBuilder Pro.
- JACK [186, 187] is a commercial agent framework by Agent Oriented Software Pty. Ltd., Melbourne, Australia. JACK is oriented towards BDI agents.
- Intelligent Agent Factory [188] by Bits & Pixels, Texas, USA.
- Grasshopper [189] is an advanced development platform for mobile agents launched by IKV++, Germany.

Related tools can be found at the IBM Aglets Development Kit homepage [190] and the Microsoft Agent homepage [191].

A comparison of four available platforms (AgentBuilder, Jack, MADKIT, and ZEUS) can be found in [192]. Some of the above mentioned platforms (e.g., JADE, ZEUS, FIPA-OS, LEAP, and Grasshopper) conform to the FIPA specifications [133]; Grasshopper is also compliant to the OMG MASIF ("Mobile Agent System Interoperability Facility") standard [193].

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 [194], and more recent examples are IMPACT [195], SWARM [196], and Agent Factory [197]. Good examples of testbeds explicitly oriented towards industrial needs and real-world applications are ARCHON [198] and MECCA [123].

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,

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

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., [199]) are abstract and reusable desciptions of solutions to particular (software) design problems. Good examples of work on design patterns for agent software are:

- In [200] a generic agent pattern format and specific patterns for agent-agent coordination are proposed.
- In [201] several patterns for agent behavior (hence also called behavior patterns), together with a development method using these patterns, are described.
- In [202] an agent design pattern for developing dynamic and distributed applications is introduced.
- In [203, 204] 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 [205] and [206] several patterns for mobile agents applications are identified.

Software architectures (e.g., [207, 208]) are structured descriptions of elements from which software systems are built. As mentioned earlier in 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:

- In [209, 210] several high-level organizational patterns for multiagent architectures are presented.
- In [211, 212] 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., [213, 214, 215]). Several of the agent-oriented methods mentioned earlier apply use cases and scenarios during analysis and design; for instance, see [59] (4.1) and [63, 66] (4.2). Other good examples of related work on agent system engineering are the following:

- In [216, 217] use case maps are applied to model and visualize overall system behavior patterns.
- In [218] an object-oriented analysis method based on use cases and IDEF functions is described.

UML ("Unified Modeling Language", e.g. [219]) 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 [220, 221, 222] an agent-oriented extension of UML, called AGENT UML or AUML, is described and illustrated. AUML is a result of cooperation among FIPA and OMG with the goal of increasing industrial acceptance of agent technology.
- In [223] 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.

[5]Another example is Z; see 4.4.

- In [224] the specification of agent interaction protocols through standard UML is proposed.
- In [225] an approach to the modelling of agents that uses UML and graph transformation is described.
- In [226] a proposal for the integration of agent roles in UML can be found.
- In [227] an approach to the UML-based modelling of multiagent architectures and ontologies for agent-agent communication is introduced.

General considerations on the requirements an ideal "unified agent-oriented modelling language" (UAML) should fulfill are provided in [105, 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 multiagent 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 on 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, e.g., 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 a rapid development and enormous progress in the 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 – still are in an early prototypical stage which is characterized 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 organization. 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 organizational, 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 standardization efforts w.r.t. agent-specific languages, interaction protocols, and specification and representational formalisms. This includes, in particular, the development of communication languages having an 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.

Acknowledgements. 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.

References

2 Basic Literature

2.1 Foundations

[1] N.R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117:277–296, 2000.

[2] N.R. Jennings and M. Wooldridge. Agent-oriented software engineering. In J. Bradshaw, editor, *Handbook of Agent Technology.* AAAI/MIT Press, 2002.

[3] M.J. Wooldridge and P. Ciancarini. Agent-oriented software engineering: the state of the art. In P. Ciancarini and M.J. Wooldridge, editors, *Agent-oriented software engineering. Proceedings of the First International Workshop (AOSE-2000)*, Lecture Notes in Artificial Intelligence, Vol. 1957, pages 1–28. Springer-Verlag, 2001.

[4] M.R. Genesereth and S.P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53 and 147 (continued), 1994.

[5] R. Gustavsson. 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, 1994.

[6] G.M.P. O'Hare and M. Wooldridge. A software engineering perspective on multi-agent system design. In N. Avouris and L. Gasser, editors, *Distributed Artificial Intelligence – Theory and Practice*, pages 109–127. Kluwer Academic Publ., Dordrecht u.a., 1992.

[7] P. Ciancarini and M. Wooldridge, editors. *Agent-oriented software engineering. Proceedings of the First International Workshop (AOSE-2000).* Lecture Notes in Computer Science, Vol. 1957. Springer-Verlag, 2001.

[8] M. Wooldridge, G. Weiß, and P. Ciancarini, editors. *Agent-oriented software engineering II. Proceedings of the Second International Workshop (AOSE-2001).* Lecture Notes in Computer Science, Vol. 2222. Springer-Verlag, 2002.

[9] F.J. Garijo and M. Boman, editors. *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 Vol. 1647. Springer-Verlag, Berlin et al., 1999.

[10] A. Omicini, R. Tolksdorf, and F. Zambonelli, editors. *Engineering Societies in the Agents World. Proceedings of the First International Workshop (ESAW 2000).* Lecture Notes in Artificial Intelligence, Vol. 1972. Springer-Verlag, Berlin et al., 2000.

2.2 Agents and Objects

[11] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In J.P. Müller, M.J. Wooldridge, and N.R. Jennings, editors, *Intelligent Agents III*, Lecture Notes in Artificial in Artificial Intelligence, Vol. 1193, pages 21–36. Springer-Verlag, Berlin et al., 1997.

[12] M.J. Wooldridge and N.R. Jennings. Agent theories, architectures, and languages: A survey. In M.J. Wooldridge and N.R. Jennings, editors, *Intelligent Agents*, Lecture Notes in Artificial in Artificial Intelligence, Vol. 890, pages 1–39. Springer-Verlag, Berlin et al., 1995.

[13] J.M. Bradshaw. An introduction to software agents. In J.M. Bradshaw, editor, *Software Agents*, pages 3–46. AAAI Press/The MIT Press, 1997.

[14] H.S. Nwana. Software agents: An overview. *The Knowledge Engineering Review*, 11(3):205–244, 1996.

[15] S.J. Russell and P. Norvig. *Artificial Intelligence. A Modern Approach.* Prentice Hall, Englewood Cliffs, New Jersey, 1995.

[16] M.J. Wooldridge. Intelligent agents. In G. Weiss, editor, *Multiagent Systems*, pages 27–77. The MIT Press, Cambridge et al., 1999.

[17] J.M. Bradshaw, editor. *Handbook of agent technology.* AAAI Press/The MIT Press, 2002.

[18] G. Weiß, editor. *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence.* The MIT Press, Cambridge, MA, 1999.

[19] M.J. Wooldridge and N.R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

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

2.3 Aspects of Assessing Agent Orientation

[21] G. Weiß. Agentenorientiertes Software Engineering. *Informatik Spektrum*, 24(2):98–101, 2001.

[22] G. Booch. *Object-Oriented Analysis and Design with applications* (2nd edition). Addison Wesley, Reading, MA, 1994.

[23] F.P. Brooks. No silver bullet. In *Proceedings of the IFIP Tenth World Computer Conference*, pages 1069–1076, 1986.

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

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

[26] M.J. Wooldridge and N.R. Jennings. Pitfalls of agent-oriented development. In *Proceedings of the Second International Conference on Autonomous Agents (Agents'98)*, pages 385–391, 1998.

[27] B. Crabtree. What chance software agents? *The Knowledge Engineering Review*, 13(2):131–136, 1998.

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

[29] C. Petrie. Agent-based software engineering. In P. Ciancarini and M.J. Wooldridge, editors, *Agent-oriented software engineering. Proceedings of the First International Workshop (AOSE-2000)*, Lecture Notes in Artificial Intelligence, Vol. 1957, pages 59–76. Springer-Verlag, 2001.

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

[31] V. Parunak. Industrial and practical applications of DAI. In G. Weiss, editor, *Multiagent Systems*, pages 377–421. The MIT Press, Cambridge et al., 1999.

[32] V. Parunak. 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, 2000.

[33] J.P. Müller. Control architectures for autonomous and interacting agents: A survey. In L. Cavedon, L. Rao, and W. Wobcke, editors, *Intelligent Agents Systems: Theoretical and Practical Issues*, Lecture Notes in Artificial in Artificial Intelligence, Vol. 1209. Springer-Verlag, Berlin et al., 1996.

[34] J.P. Müller. The right agent (architecture) to do the right thing. In J.P. Müller, M.P. Singh, and A.S. Rao, editors, *Intelligent Agents V*, Lecture Notes in Artificial in Artificial Intelligence, Vol. 1555, pages 211–226. Springer-Verlag, Berlin et al., 1999.

[35] K.S. Decker, E.H. Durfee, and V.R. Lesser. Evaluating research in cooperative distributed problem solving. In M.N. Huhns and L. Gasser, editors, *Distributed Artificial Intelligence, Volume 2*, pages 487–519. Pitman/Morgan Kaufmann, Cambridge, MA, 1989.

3 Methods and Frameworks for Requirements Engineering

[36] B.A. Nuseibeh and S.M. Easterbrook. Requirements engineering: A roadmap. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE'00)*, 2000.

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

[38] A. van Lamsweerde. Requirements engineering in the year 00: A research perspective. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE'00)*, 2000.

[39] E.S.K. Yu. Why agent-oriented requirements engineering? In *Proceedings of 3rd International Workshop on Requirements Engineering: Foundations for Software Quality*, 1997.

[40] E.S.K. Yu. Agent orientation as a modelling paradigm. *Wirtschaftsinformatik*, 43(2):123–132, 2001.

[41] E.S.K. Yu. Agent-oriented modelling: software versus the world. In M.J. Wooldridge, G. Weiß, and P. Ciancarini, editors, *Agent-oriented software engineering. Proceedings of the Second International Workshop (AOSE-2001)*, Lecture Notes in Artificial Intelligence, Vol. 2222. Springer-Verlag, 2002.

[42] E.S.K. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of 3rd IEEE International Symposium on Requirements Engineering (RE'97)*, pages 226–235, 1997.

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

[44] P. Du Bois. The ALBERT II reference manual. Technical Report RR-97-002, Computer Science Department, University of Namur, Belgium, 1997.

[45] E.S.K. Yu, P. Du Bois, E. Dubois, and J. Mylopoulos. From organization models to system requirements – A "cooperating agents" approach. In *Proceedings of the Third International Conference on Cooperative Information Systems (CoopIS'95)*, pages 194–204, 1995.

[46] J. Mylopoulos and J. Castro. Tropos: A framework for requirements-driven software development. In J. Brinkkemper and A. Solvberg, editors, *Information systems engineering: State of the art and research themes*. Springer-Verlag, 2000.

[47] J. Mylopoulos, L. Chung, and E.S.K. Yu. From object-oriented to goal-oriented requirements analysis. *Communications of the ACM*, 42(1):31–37, 1999.

[48] E.S.K. Yu and J. Mylopoulos. Why goal-oriented requirements engineering? In *Proceedings of the 4th International Workshop on Requirements Engineering*, pages 15–22, 1998.

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

[50] L. Chung, B.A. Nixon, E. Yu, and J. Mylopoulos. *Non-functional requirements in software engineering*. Kluwer Academic Press, Boston et al., 2000.

4 Methods and Frameworks for Analysis, Design, and Implementation

4.1 Approaches based on Agent and Multiagent Technology

[51] M.J. Wooldridge, N.R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[52] F. Zambonelli, N.R. Jennings, A. Omicini, and M. Wooldridge. Agent-oriented software engineering for Internet applications. In A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, editors, *Coordination of Internet Agents: Models, Technologies and Applications*. Springer-Verlag, Berlin et al., 2000.

[53] A. Omicini. SODA: Societies and infrastructures in the analysis and design of agent-based systems. In P. Ciancarini and M.J. Wooldridge, editors, *Agent-oriented software engineering. Proceedings of the First International Workshop (AOSE-2000)*, Lecture Notes in Artificial Intelligence, Vol. 1957, pages 185–194. Springer-Verlag, 2001.

[54] A. Drogoul and A. Collinot. 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, 1998.

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

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

[57] W. Brauer, M. Nickles, M. Rovatsos, G. Weiß, and K.F. Lorentzen. Expectation-oriented analysis and design. In M.J. Wooldridge, G. Weiß, and P. Ciancarini, editors, *Agent-oriented software engineering. Proceedings of the Second International Workshop (AOSE-2001)*, Lecture Notes in Artificial Intelligence, Vol. 2222. Springer-Verlag, 2002.

[58] V. Dignum, H. Weigand, and L. Xu. Agent societies: Toward frameworks-based design. In M.J. Wooldridge, G. Weiß, and P. Ciancarini, editors, *Agent-oriented software engineering. Proceedings of the Second International Workshop (AOSE-2001)*, Lecture Notes in Artificial Intelligence, Vol. 2222. Springer-Verlag, 2002.

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

[60] F. Zambonelli, N.R. Jennings, and M. Wooldridge. Organisational abstractions for the analysis and design of multi-agent systems. In P. Ciancarini and M.J. Wooldridge, editors, *Agent-oriented software engineering. Proceedings of the First International Workshop (AOSE-2000)*, Lecture Notes in Artificial Intelligence, Vol. 1957, pages 235–252. Springer-Verlag, 2001.

4.2 Approaches based on Object-Oriented Technology

[61] D. Kinny, M. Georgeff, and A. Rao. A methodology and modelling technique for systems of BDI agents. In W. van der Velde and J. Perram, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-96)*, Lecture Notes in Artificial Intelligence Vol. 1038, pages 56–71. Springer-Verlag, 1996.

[62] M.P. Georgeff and A.S. Rao. BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319, 1995.

[63] M. Wood and S.A. DeLoach. An overview of the multiagent systems engineering methodology. In P. Ciancarini and M.J. Wooldridge, editors, *Agent-oriented software engineering. Proceedings of the First International Workshop (AOSE-2000)*, Lecture Notes in Artificial Intelligence, Vol. 1957, pages 207–222. Springer-Verlag, 2001.

[64] J. Lind. *Iterative software engineering for multiagent systems: The MASSIVE method*. Lecture Notes in Computer Science, Vol. 1994. Springer-Verlag, Berlin u.a., 2001.

[65] B. Burmeister. Models and methodology for agent-oriented analysis and design. In K. Fischer, editor, *Working Notes of the KI96 Workshop on Agent-oriented Programming and Distributed Systems*. DFKI Dokument D-96-06, 1996.

[66] B. Moulin and M. Brassad. A scenario-based design method and an environment for the development of multiagent systems. In D. Luckose and C. Zhang, editors, *Proceedings of the First Australian Workshop on DAI*, Lecture Notes in Artificial Intelligence, pages 216–296. Springer-Verlag, 1996.

[67] E.A. Kendall. Agent roles and role models: New abstractions for multiagent system analysis and design. In *International Workshop on Intelligent Agents in Information and Process Management*, 1998.

[68] G. Wagner. 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, 2000.

[69] J. Bryson and B. McGonigle. Agent architecture as object oriented design. In M.P. Singh, A. Rao, and M.J. Wooldridge, editors, *Intelligent Agents IV. Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL-97)*, Lecture Notes in Artificial Intelligence Vol. 1365, pages 15–30. Springer-Verlag, 1998.

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

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

[72] G. Satapathy and S.R.T. Kumara. Object oriented design based agent modeling. In *Proceedings of the Fourth International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'99)*, pages 143–162, 1999.

4.3 Approaches based on Knowledge Engineering Technology

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

[74] N. Glaser. *Contribution to Knowledge Modelling in a Multi-agent Framework*. PhD thesis, Université Henry Poincaré, Nancy, France, 1996.

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

[76] C. Iglesias, M. Garijo, J.C. Gonzales, and J.R. Velasco. Analysis and design of multi-agent systems using MAS-CommonKADS. In M.P. Singh, A. Rao, and M.J. Wooldridge, editors, *Intelligent Agents IV. Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL-97)*, Lecture Notes in Artificial Intelligence Vol. 1365, pages 313–326. Springer-Verlag, 1998.

[77] R.E. Gustavsson. Multi agent systems as open societies – A design framework. In M.P. Singh, A. Rao, and M.J. Wooldridge, editors, *Intelligent Agents IV. Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL-97)*, Lecture Notes in Artificial Intelligence Vol. 1365, pages 329–337. Springer-Verlag, 1998.

4.4 Formal Specification and Verification

[78] M.J. Wooldridge. Agents and software engineering. *AI*IA Notizie*, XI(3):31–37, September 1998.

[79] P.R. Cohen and H.J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.

[80] A.S. Rao and M.P. Georgeff. BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319, 1995.

[81] M.J. Wooldridge, editor. *Reasoning About Rational Agents.* The MIT Press, Cambridge, MA, 2000.

[82] M.P. Singh, M.N. Huhns, and L.M. Stephens. Declarative representations of multiagent systems. *IEEE TKDE*, 5(5):721–739, 1993.

[83] F.M.T. Brazier, B.M. Dunin-Keplicz, N.R. Jennings, and J. Treur. DESIRE: Modelling multi-agent systems in a compositional framework. *International Journal of Cooperative Information Systems*, 6(1):67–94, 1997.

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

[85] S.J. Rosenschein and L.P. Kaelbling. The synthesis of digital machines with provable epistemic properties. In *Proceedings of the Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 83–98, 1986.

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

[87] M.J. Wooldridge. *The Logical Modelling of computational Multi-Agent Systems.* PhD thesis, Department of Computation, UMIST, Manchester, UK, 1992.

[88] K. Schild. On the relationship between BDI logics and standard logics of concurrency. In J.P. Müller, M.P. Singh, and A. Rao, editors, *Intelligent Agents V. Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98)*, Lecture Notes in Artificial Intelligence Vol. 1555, pages 47–61. Springer-Verlag, 1999.

[89] M. Benerecetti, F. Giunchiglia, and L. Serafini. A model checking algorithm for multi-agent systems. In J.P. Müller, M.P. Singh, and A. Rao, editors, *Intelligent Agents V. Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98)*, Lecture Notes in Artificial Intelligence Vol. 1555, pages 163–176. Springer-Verlag, 1999.

[90] A.S. Rao and M.P. Georgeff. A model-theoretic approach to the verification of situated reasoning systems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, 1993.

[91] J.M. Spivey. *The Z Notation* 2nd edition). Prentice Hall, Hempstead, 1992.

[92] M. d'Inverno and M. Luck, editors. *Understanding agent systems.* Springer-Verlag, 2001.

[93] M. Fisher. If Z is the answer, what could the question possibly be? In J.P. Müller, M.J. Wooldridge, and N.R. Jennings, editors, *Intelligent Agents III*, Lecture Notes in Artificial in Artificial Intelligence, Vol. 1193, pages 65–69. Springer-Verlag, Berlin et al., 1997.

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

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

[96] Z. Manna and A. Pnueli, editors. *Temporal Verification of Reactive Systems.* Springer-Verlag, Berlin u.a., 1995.

[97] A. Pnueli. Specification and development of reactive systems. In *Information Processing 86.* Elsevier Science Publ., 1986.

[98] A.W. Roscoe, editor. *Theory and practice of concurrency.* Prentice Hall, 1997.

[99] R.A. Dooley. Appendix B: Repartee as a graph. In R.E. Longacre, editor, *An anatomy of speech notions*, pages 348–358. Peter de Ridder: Lisse, Holland, 1976.

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

[101] M.P. Singh. Synthesizing coordination requirements for heterogeneous autonomous agents. *Autonomous Agents and Multi-Agent Systems*, 3(2):107–132, 2000.

[102] M. d'Inverno, M. Fisher, A. Lomuscio, M. Luck, M. de Rijke, M. Ryan, and M. Wooldridge. Formalisms for multi-agent systems. *The Knowledge Engineering Review*, 3(12), 1997.

4.5 Further Key Issues

[103] EURESCOM/MESSAGE. 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, July 2000.

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

[105] O. Arazy and C.C. Woo. Analysis and design of agent-oriented information systems. Working Paper 99-MIS-004, Faculty of Commerce and Business Administration, University of British Columbia, Canada, 2000.

[106] C. Iglesias, M. Garijo, and J.C. Gonzales. A survey of agent-oriented methodologies. In J.P. Müller, M.P. Singh, and A. Rao, editors, *Intelligent Agents V. Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98)*, Lecture Notes in Artificial Intelligence Vol. 1555, pages 317–330. Springer-Verlag, 1999.

5 Languages

5.1 Programming Languages

[107] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[108] M.P. Wellman. Market-oriented programming: Some early lessons. In S.H. Clearwater, editor, *Market-based control*, pages 74–95. World Scientific, 1996.

[109] M.P. Singh. Toward interaction-oriented programming. In *Proceedings of the 2nd International Conference on Multi-Agent Systems (ICMAS-96)*, page 457, 1996.

[110] M.N. Huhns. Interaction-oriented programming. In P. Ciancarini and M.J. Wooldridge, editors, *Agent-oriented software engineering. Proceedings of the First International Workshop (AOSE-2000)*, Lecture Notes in Artificial Intelligence, Vol. 1957, pages 29–44. Springer-Verlag, 2001.

[111] S.R. Thomas. The PLACA agent programming language. In M.J. Wooldridge and N.R. Jennings, editors, *Intelligent Agents*, Lecture Notes in Artificial in Artificial Intelligence, Vol. 890, pages 355–370. Springer-Verlag, Berlin et al., 1995.

[112] W.H.E. Davies and P. Edwards. AGENT-K: An integration of AOP and KQML. In *Proceedings of the CIKM'94 Workshop on Intelligent Agents*, 1994.

[113] M. Fisher. Representing and executing agent-based systems. In M.J. Wooldridge and N.R. Jennings, editors, *Intelligent Agents*, Lecture Notes in Artificial in Artificial Intelligence, Vol. 890, pages 307–323. Springer-Verlag, Berlin et al., 1995.

[114] M. Mulder, J. Treur, and M. Fisher. Agent modelling in METATEM and DESIRE. In M.P. Singh, A. Rao, and M.J. Wooldridge, editors, *Intelligent Agents IV. Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL-97)*, Lecture Notes in Artificial Intelligence Vol. 1365, pages 193–208. Springer-Verlag, 1998.

[115] A.S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. van der Velde and J. Perram, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-96)*, Lecture Notes in Artificial Intelligence Vol. 1038, pages 42–55. Springer-Verlag, 1996.

[116] M.P. Georgeff and A.L. Lansky. Reactive reasoning and planning. In *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-87)*, pages 677–682, 1987.

[117] M. d'Inverno and M. Luck. Engineering AgentSpeak(L): A formal computational model. *Journal of Logic and Computation*, 8(3):233–260, 1998.

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

[119] K.v. Hindriks, F.S. de Boer, W. van der Hoek, and J.-J. Meyer. A formal embedding of AgentSpeak(L) in 3APL. In G. Antoniou and J. Slaney, editors, *Advanced Topics in Artificial Intelligence*, Lecture Notes in Artificial Intelligence, Vol. 1502, pages 155–166. Springer-Verlag, Berlin et al., 1998.

[120] G. De Giacomo, Y. Lespérance, and H. Levesque. ConGolog: A concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121:109–169, 2000.

[121] K.v. Hindriks, Y. Lespérance, and H.J. Levesque. A formal embedding of ConGolog in 3APL. In *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI-2000)*, pages 558–562, 2000.

[122] F.G. McCabe and K.L. Clark. April – Agent PRocess Interaction Language. In M.J. Wooldridge and N.R. Jennings, editors, *Intelligent Agents*, Lecture Notes in Artificial in Artificial Intelligence, Vol. 890, pages 324–340. Springer-Verlag, Berlin et al., 1995.

[123] D.D. Steiner. IMAGINE: An integrated environment for constructing distributed artificial intelligence systems. In G.M.P. O'Hare and N.R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 345–364. Wiley, New York et al., 1996.

[124] G. Wagner. VIVA knowledge-based agent programming. Preprint, Institut für Informatik, Universität Leipzig, Germany, 1996.

[125] A. Poggi. Daisy: An object-oriented system for distributed artificial intelligence. In M.J. Wooldridge and N.R. Jennings, editors, *Intelligent Agents*, Lecture Notes in Artificial Intelligence, Vol. 890, pages 341–354. Springer-Verlag, Berlin et al., 1995.

[126] J.E. White. Mobile agents. In J.M. Bradshaw, editor, *Software Agents*, pages 437–472. AAAI Press/The MIT Press, 1997.

[127] H.D. Burkhard. Agent-oriented programming in open systems. In M.J. Wooldridge and N.R. Jennings, editors, *Intelligent Agents*, Lecture Notes in Artificial Intelligence, Vol. 890, pages 291–306. Springer-Verlag, Berlin et al., 1995.

5.2 Languages for Communication and Coordination

[128] T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In J.M. Bradshaw, editor, *Software Agents*, pages 291–316. AAAI Press/The MIT Press, 1997.

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

[130] R.S. Cost, I. Soboroff, J. Lakhani, T. Finin, E. Miller, and C. Nicholas. TKQML: A scripting tool for building agents. In M.P. Singh, A. Rao, and M.J. Wooldridge, editors, *Intelligent Agents IV. Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL-97)*, Lecture Notes in Artificial Intelligence Vol. 1365, pages 339–343. Springer-Verlag, 1998.

[131] M.D. Sadek. Dialogue acts are rational plans. In *Proceedings of the ESCA/ETRW Workshop on the Structure of multimodal Dialogue*, pages 1–29, 1991.

[132] M.D. Sadek, P. Bretier, and F. Panaget. Artimis: natural dialogue meets rational agency. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, 1997.

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

[134] M.R. Genesereth and R.E. Fikes. Knowledge Interchange Format. Version 3.0, Reference Manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, 1992.

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

[136] M. Barbuceanu and M.S. Fox. Capturing and modeling coordination knowledge for multiagent systems. *International Journal of Cooperative Information Systems*, 5(2-3):275–314, 1996.

[137] D.L. Martin, A.J. Cheyer, and D.B. Moran. The open agent architecture: a framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1/2):91–128, 1999.

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

[139] E.M. Verharen, F. Dignum, and S. Bos. Implementation of a cooperative agent architecture based on the language-action perspective. In M.P. Singh, A. Rao, and M.J. Wooldridge, editors, *Intelligent Agents IV. Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL-97)*, Lecture Notes in Artificial Intelligence Vol. 1365, pages 31–44. Springer-Verlag, 1998.

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

[141] M. Cremonini, A. Omicini, and F. Zambonelli. Multi-agent systems on the Internet: Extending the scope of coordination towards security and topology. In F.J. Garijo and M. Boman, editors, *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 Vol. 1647, pages 77–88. Springer-Verlag, 1999.

[142] E. Denti and A. Omicini. Engineering multi-agent systems in LuCe. In Stephen Rochefort, Fariba Sadri, and Francesca Toni, editors, *Proceedings of the ICLP'99 International Workshop on Multi-Agent Systems in Logic Programming (MAS'99)*, Las Cruces (NM), 1999.

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

[144] M. Schumacher, F. Chantemargue, and B. Hirsbrunner. The STL++ coordination language: A base for implementing distributed multi-agent applications. In P. Ciancarini and A.. Wolf, editors, *Proceedings of the Third International Conference on Coordination Languages and Models (COORDINATION'99)*. Springer-Verlag, 1999.

[145] S. Moss, H. Gaylard, S. Wallis, and B. Edmonds. SDML: A multi-agent language for organizational modelling. CPM Report 97-19, Centre for Policy Modelling, Manchester Metropolitan University, United Kingdom, 1996.

[146] M.P. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):55–61, 1998.

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

[148] M.J. Wooldridge. Verifiable semantics for agent communication languages. In *Proceedings of the 3nd International Conference on Multi-Agent Systems (ICMAS-98)*, pages 349–356, 1998.

[149] F. Dignum. Agent communication and cooperative information agents. In M. Klusch and L. Kerschberg, editors, *Cooperative Information Agents IV. Proceedings of the Fourth International Workshop on Cooperative Information Agents (CIA-2000)*, Lecture Notes in Artificial in Artificial Intelligence, Vol. 1860, pages 191–207, Berlin et al., 2000. Springer-Verlag.

[150] M.T. Kone, A. Shimazu, and T. Nakajima. The state of the art in agent communication languages. *Knowledge and Information Systems*, 2:259–284, 2000.

[151] F. Arbab, P. Ciancarini, and C. Hankin. Coordination languages for parallel programming. *Parallel Computing*, 1998.

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

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

[154] G.A. Papadopoulos and F. Arbab. Coordination models and languages. *Advances in Computers*, 46:329–400, 1998.

[155] A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, editors. *Coordination of Internet Agents: Models, Technologies, Applications.* Springer-Verlag, Berlin et al., 2000.

5.3 Ontology Specification Languages

[156] T.R. Gruber. Ontolingua: A mechanism to support portable ontologies. Technical Report KSL-91-66, Knowledge Systems Laboratory, Stanford University, 1992.

[157] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42, 1995.

[158] A. Bordiga, R.J. Brachman, D.L. McGuinness, and L.A. Resnick. CLASSIC: A structural data model for objects. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 59–67, 1989.

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

[160] D.B. Lenat and R.V. Guha, editors. *Building large knowledge-based systems. Representation and inference in the Cyc project.* Addison-Wesley, Reading, MA, 1990.

[161] SHOE. http://www.cs.umd.edu/projects/plus/shoe/, 2001.

[162] XOL. http://www.ontologos.org/ontology/xol.htm, 2001.

[163] OIL. http://www.ontoknowledge.org/oil/, 2001.

[164] DAML-LANGUAGE. http://www.daml.org/language/, 2001.

[165] PROTEGE. http://protege.stanford.edu/index.shtml, 2001.

[166] WEBONTO. http://webonto.open.ac.uk, 2001.

[167] ONTOEDIT. http://www.ontoprise.com, 2001.

[168] D. Fensel. *Ontologies: a silver bullet for knowledge management and electronic commerce.* Springer-Verlag, 2001.

[169] T.R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 41:399–424, 1995.

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

6 Development Tools and Platforms

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

[172] D.T. Ndumu, H.S. Nwana, L. Lee, and J. Collins. Visualising and debugging distributed multi-agent systems. In *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 326–333, 1999.

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

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

[175] agenTool. http://www.cis.ksu.edu/˜sdeloach/ai/agentool.htm, 2000.

[176] RETSINA. http://www-2.cs.cmu.edu/˜softagents/, 2000.

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

[178] FIPA-OS. http://fipa-os.sourceforge.net/, 2000.

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

[180] SIM_AGENT. http://www.cs.bham.ac.uk/˜axs/cog_affect/sim_agent.html, 1996.

[181] JAFMAS. http://www.ececs.uc.edu/˜abaker/jafmas/, 2000.

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

[183] OAA. *Open Agent Architecture*, http://www.ai.sri.com/˜oaa/, 1999.

[184] D. Kinny. The Agentis agent interaction model. In J.P. Müller, M.P. Singh, and A. Rao, editors, *Intelligent Agents V. Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98)*, Lecture Notes in Artificial Intelligence Vol. 1555, pages 331–344. Springer-Verlag, 1999.

[185] AgentBuilder. http://www.agentbuilder.com/, 1999.

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

[187] JACK. *JACK Intelligent Agents*, ttp://www.agent-software.com.au/shared/home/index.html, 1998.

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

[189] Grasshopper. http://www.grasshopper.de/index.html, 1998.

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

[191] Microsoft Agent. http://msdn.microsoft.com/ library/ default.asp?url=/library/ en-us/ msagent/ agentstartpage_7gdh.asp, 2000.

[192] P.-M. Ricordel and Y. Demazeau. From analysis to deployment: A multi-agent platform survey. In *Working Notes of the First International Workshop on Engineering Societies in the Agents' World (ESAW-00)*, 2000.

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

[194] K.S. Decker. Distributed artificial intelligence testbeds. In G.M.P. O'Hare and N.R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 119–138. John Wiley & Sons Inc., New York, 1996.

[195] V.S. Subrahmanian, P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Ozcan, and R. Ross. *Heterogeneous agent systems.* The MIT Press, Cambridge et al., 2000.

[196] Swarm. *Swarm Development Group*, http://www.swarm.org/, 2000.

[197] G.M.P. O'Hare. Agent factory: An environment for the fabrication of multiagent systems. In G.M.P. O'Hare and N.R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 449–484. Wiley, New York et al., 1996.

[198] D. Cockburn and N.R. Jennings. ARCHON: A distributed artificial intelligence system for industrial applications. In G.M.P. O'Hare and N.R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 319–344. Wiley, New York et al., 1996.

7 Other Approaches

[199] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, editors. *Design patterns.* Addison-Wesley, Reading, MA, 1995.

[200] D. Deugo, M. Weiss, and E. Kendall. Reusable patterns for agent coordination. In A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, editors, *Coordination of Internet Agents: Models, Technologies and Applications*, pages 347–368. Springer-Verlag, Berlin et al., 2000.

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

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

[203] E.A. Kendall and M.T. Malkoun. The layered agent patterns. In *Pattern Languages of Programs (PLoP'96)*, 1996.

[204] E.A. Kendall, C.V. Pathak, P.V.M. Krishna, and C.B. Suresh. The layered agent pattern language. In *Proceedings of the Conference on Pattern Languages of Programs (PLoP'97)*, 1997.

[205] Y. Aridor and D.B. Lange. Agent design patterns: Elements of agent application design. In *Proceedings of the Second International Conference on Autonomous Agents (Agents'98)*, pages 108–115, 1998.

[206] D. Deugo, F. Oppacher, J. Kuester, and I.V. Otte. Patterns as a means for intelligent software engineering. In *Proceedings of the International Conference on Artificial Intelligence (IC-AI'99)*, volume II, pages 605–611, 1999.

[207] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-oriented software architecture. A system of patterns.* Wiley, New York, 1996.

[208] M. Shaw and D. Garlan. *Software architecture: Perspectives on an emerging discipline.* Prentice Hall, 1996.

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

[210] M. Kolp, P. Giorgini, and J. Mylopoulos. An goal-based organizational perspective on multi-agents architectures. In *Proceedings of the Eighth International Workshop on Agent Theories, architectures, and languages (ATAL-2001)*, 2001.

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

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

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

[214] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object-oriented software engineering. A use case driven approach.* Addison-Wesley, 1994 (Revised Printing).

[215] K. Weidenhaupt, M. Pohl, M. Jarke, and P. Haumer. Scenario usage in system development: A report on current practice. *IEEE Software*, pages 34–45, 1998.

[216] R. Buhr, D. Amyot, M. Elammari, D. Quesnel, T. Gray, and S. Mankovski. 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)*, pages 277–298, 1998.

[217] R. Buhr, M. Elammari, T. Gray, and S. Mankovski. Applying use case maps to multi-agent systems: A feature interaction example. In *Hawaii International Conference on System Sciences (HICSS'98)*, 1998.

[218] E.A. Kendall, M.T. Malkoun, and C. Jiang. The application of object-oriented analysis to agent based systems, 1997.

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

[220] B. Bauer. UML class diagrams revisited in the context of agent-based systems. In M.J. Wooldridge, G. Weiß, and P. Ciancarini, editors, *Agent-oriented software engineering. Proceedings of the Second International Workshop (AOSE-2001)*, Lecture Notes in Artificial Intelligence, Vol. 2222. Springer-Verlag, 2002.

[221] B. Bauer, J.P. Müller, and J. Odell. Agent UML: A formalism for specifying multiagent software systems. In P. Ciancarini and M.J. Wooldridge, editors, *Agent-oriented software engineering. Proceedings of the First International Workshop (AOSE-2000)*, Lecture Notes in Artificial Intelligence, Vol. 1957, pages 91–103. Springer-Verlag, 2001.

[222] J. Odell, V. Parunak, and B. Bauer. Representing agent interaction protocols in UML. In P. Ciancarini and M.J. Wooldridge, editors, *Agent-oriented software engineering. Proceedings of the First International Workshop (AOSE-2000)*, Lecture Notes in Artificial Intelligence, Vol. 1957, pages 121–140. Springer-Verlag, 2001.

[223] V. Parunak and J. Odell. Representing social structures in UML. In M.J. Wooldridge, G. Weiß, and P. Ciancarini, editors, *Agent-oriented software engineering. Proceedings of the Second International Workshop (AOSE-2001)*, Lecture Notes in Artificial Intelligence, Vol. 2222. Springer-Verlag, 2002.

[224] J. Lind. Specifying agent interaction protocols with standard UML. In M.J. Wooldridge, G. Weiß, and P. Ciancarini, editors, *Agent-oriented software engineering. Proceedings of the Second International Workshop (AOSE-2001)*, Lecture Notes in Artificial Intelligence, Vol. 2222. Springer-Verlag, 2001.

[225] R. Depke, R. Heckel, and J.M. Küster. Improving the agent-oriented modelling process with roles. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents'01)*, pages 640–647, 2001.

[226] R. Depke, G. Engels, and J.M. Küster. On the integration of roles in UML. Technical Report No. 214, University of Paderborn, Germany, 2000.

[227] F. Bergenti and A. Poggi. A development environment for the realization of open and scalable multi-agent systems. In F.J. Garijo and M. Boman, editors, *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 Vol. 1647, pages 52–62. Springer-Verlag, 1999.