Chapter I

Agent-Oriented Methodologies: An Introduction

Paolo Giorgini
University of Trento, Italy

Brian Henderson-Sellers
University of Technology, Sydney, Australia

Abstract

As an introduction to agent-oriented (AO) methodologies, we first describe the characteristics of both agents and multi-agent systems (MASs). This leads to a discussion of what makes an AO methodology that can be used to build an MAS. Finally, we briefly introduce the ten methodologies that are described in the remaining chapters in this book.

Introduction

A methodology aims to prescribe all the elements necessary for the development of a software system, especially in the context of commercial applications. Prior to industry adoption, however, it is necessary for researchers to create that methodology. This has led to academic and industry researchers creating a large number of methodological approaches. A decade ago, there were estimated to

be over a thousand methodological approaches to software development (Jayaratna, 1994), although these can be grouped into a much smaller number (around five) of software development *approaches* (Iivari, Hirschheim, & Klein, 1999). To these can be added a sixth: agent-oriented (AO) methodologies; that is, methodological approaches suitable for the development of agent-oriented or agent-based software.[1]

In parallel to the growth and availability of object-oriented (OO) systems development methodologies in the nineties, we are now seeing the burgeoning of a number of innovative AO methodologies, several of which form the core of this book. However, in contrast to OO methodologies, the field is not industry-driven—most AO methodologies are supported by small teams of academic researchers. Based on an observation that the coalescence of groups of OO methodologies in the late 1990s led to an increased take-up by industry of the object-oriented paradigm for system development and project management, this book aims to encourage first the coalescence and collaboration between research groups and then, hopefully, more rapid industry adoption of AO methodological approaches. In other words, most AO methodologies are (at the time of writing) in an early stage and still in the first context of mostly "academic" methodologies for agent-oriented systems development, albeit that many of these methodologies have been tested in small, industrial applications. One purpose of this book is to identify those predominant and tested AO methodologies, characterize them, analyse them, and seek some method of unification and consolidation with the hope that, in so doing, the community of scholars supporting AO methodologies will soon be able to transfer those innovative ideas into industry acceptance. This means mimicking the OO transition curve by seeking consolidation. One means of such consolidation is discussed in the last chapter of the book: the use of a method engineering framework (e.g., Martin & Odell, 1995) to create a repository of agent-oriented method fragments.

Agents and Multi-Agent Systems

Defining agents is not straightforward. There are many opinions, some of which you will see reflected in later chapters of this book (see also discussions in, for example, Luck, Ashri & D'Inverno, 2004). The key characteristics of agents are widely understood to be highly autonomous, proactive, situated, and directed software entities. Other characteristics such as mobility are optional and create a special subtype of agent; whereas some characteristics cannot be used as determining factors since they are really grey shades of a scale that encompasses both objects and agents.

In this context, an autonomous agent is one that is totally independent and can decide its own behaviour, particularly how it will respond to incoming communications from other agents. A proactive agent is one that can act without any external prompts. However, it should be noted that this introduces some problems, since there is also a large literature on purely *reactive* agents that would not be classified as agents with this categorization. Although reactive agents dominate in some domains, in reality, most agents being designed today have both proactive and reactive behaviour. Balancing the two is the key challenge for designers of agent-oriented software systems.

The characteristic of situatedness means that agents are contained totally within some specific environment. They are able to perceive this environment, be acted upon by the environment, and, in turn, affect the environment. Finally, the directedness means that agents possess some well-defined goal and their behaviour is seen as being directed towards effecting or achieving that goal.

Comparison with objects is often made. Some see agents as "clever objects" or "objects that can say no." This means that a hybrid agent+object system is entirely feasible. Others see agents at a much higher level of abstraction (e.g., Milgrom et al., 2001), much in the same way that OO specialists view components at a similar more granular level. Indeed, it is still unresolved as to how the scale of objects, components, and agents are matched and to what extent hybrid object/component/agent systems are feasible.

Some consequences of these high-level definitions are that agents participate in decision-making cycles, sometimes labelled as "perceive-decide-act" cycles. To achieve this, we have to consider other lower-level characteristics such as the roles that agents play, the metaphor of the agents having a mental state, including the possession of skills and responsibilities, aptitudes, and capabilities. When considering their interactions via perceptions and actions with other agents and the environment, we introduce notions of perceptions, actions, and agent communication languages. Negotiating skills involve the consideration of contract nets, auction strategies, and the issues of competition versus cooperation.

Defining a Multi-Agent System (MAS) is also not straightforward. However, almost all the definitions given in the literature conceive a MAS as a system composed of cooperative or competitive agents that interact with one another in order to achieve individual or common goals. From the software engineering point of view, one of the most important characteristics of a MAS is that the final set of agents is generally not given at design time (only the initial set is specified), but rather at run time. This basically means that, in practice, MASs are based on open architectures that allow new agents to dynamically join and leave the system. The major difference with the OO approach, where objects can also be given at run time and join and leave the system dynamically, is that agents can do this autonomously showing proactive behaviors not completely predictable a priori.

A MAS contains agents (and not, say, objects); consequently, it too has some of these typical agent characteristics. Thus key characteristics of a MAS can be said to be autonomy, situatedness, proactivity, and sociality. Of these, perhaps proactivity is the most contentious since, as noted above, it is generally agreed that agents possess various degrees of both proactive and reactive behaviour. Secondly, autonomy is not a binary characteristic either, since active objects, used for instance in the context of event-driven programming, can also be said to exhibit *some* degree of proactivity. Notwithstanding these two concerns, these agent characteristics lead to a set of desirable high-level characteristics (Milgrom et al., 2001) including adaptiveness, flexibility, scalability, maintainability, and likely emergent behaviour. While these are said to be "desirable" characteristics, it is perhaps the last of this list that causes most concern. Emergence is usually linked to Complex Adaptive Systems (CAS) theory. Although there are many shades of definition of what is meant by emergence in the CAS community, the general interpretation is that an emergent behaviour is one that cannot be predicted by inspection of the individual parts. This means that it is not visible from a bottom-up analysis and, arguably therefore, must be considered at the system level. Since this sort of emergent behaviour is generally encouraged, allowing it to emerge unconstrained and unplanned is clearly dangerous in certain circumstances (whilst beneficial in others). To alleviate this concern of an uncontrolled and uncontrollable agent system wreaking havoc, clearly emergent behaviour has to be considered and planned for at the systems level using top-down analysis and design techniques. This is still an area that is largely unknown in MAS methodologies. ADELFE starts along this path with its consideration of adaptive MAS (see Chapter VII for further details) in which agents that permanently try to maintain cooperative interactions with others.

Many AO methodologies (e.g., Gaia and Tropos) use the metaphor of the human organization (possibly divided into sub-organizations) in which agents play one or more roles and interact with each other. Human organization models and structures are consequently used to design MAS (see, for instance, the use of architectural patterns in Tropos or the organization models in MAS-commonKADS). Concepts like role, social dependency, and organizational rules are used not just to model the environment in which the system will work, but the system itself. Given the organizational nature of a MAS, one of the most important activities in an AO methodology results in the definition of the interaction and cooperation models that capture the social relationships and dependencies between agents and the roles they play within the system. Interaction and cooperation models are generally very abstract, and they are concretized implementing interaction protocols in later phases of the design.

Although the Agent-Oriented Programming (AOP) paradigm was introduced more than ten years ago by Yoav Shoam in his seminal work (Shoham, 1993), still there are no AO languages used in practice for developing an MAS. Some tools

have been developed in recent years to support the implementation of agents and multi-agent systems, but still none are based on a proper agent-oriented language. An interesting and very long list of agent tools is available at the AgentLink Web site (http://www.agentlink.org). Current agent development tools are mainly built on top of Java and use the object-oriented paradigm for implementing software. The methodologies presented in this book do not have as their main focus the implementation phase, although many of them give some indications how to do that. The most used developing tools are JACK (AOS, 2000) and JADE (*http://www.jade.cselt.it/).*

JACK is a commercial agent-oriented development environment built on top of and fully integrated with Java. It includes all components of the Java development environment and also offers specific extensions to implement agent behaviour. JACK provides agent-oriented extensions to the Java programming language whereby source code is first compiled into regular Java code before being executed. In JACK, a system is modelled in terms of agents defined by capabilities, which in turn are defined in terms of plans (set of actions), events, beliefs, and other capabilities.

JADE (Java Agent DEvelopment Framework) is a free software framework fully implemented in the Java language. It allows for the implementation of multi-agent systems through middleware that complies with the FIPA specifications and through a set of graphical tools that supports the debugging and deployment phases. The agent platform can be distributed across machines (which do not even need to share the same operating system) and the configuration can be controlled via a remote GUI. The configuration can even be changed at run-time by moving agents from one machine to another one, as and when required.

AO Methodologies

What is an AO Methodology?

While there is much debate on the use of terminology in various subcultures of information systems and software engineering, it can be generally agreed (e.g., Rolland, Prakash, & Benjamen, 1999) that a "methodology" has two important components: one that describes the process elements of the approach, and a second that focuses on the work products and their documentation. The second of these is more visible in the usage of a methodology, which is why the OO modelling language UML (OMG, 2001) is so frequently (and totally incorrectly) equated with "all things OO" or even described as a methodology! A modelling language such as this or its agent-focused counterpart of AUML (Odell,

Parunak, & Bauer, 2000) offers an important contribution but has limited scope within the context of a methodology. The emphasis placed on the product side of a methodology tends to vary between different authors, as will be seen in the remainder of this book. Some use UML/AUML and others eschew this as being inadequate to support the concepts of agents as described in our earlier section on Agent and Multi-agent Systems, introducing instead their own individualistic notation and underpinning concepts. When UML-style diagrams are used, it is assumed that the reader has some familiarity with this kind of graphical notation[2]; otherwise, each author fully defines the notational set of icons being used in that particular methodological approach.

Although these two main components (process and product support) are generically agreed upon, it is possible to elaborate a little more since there are issues of people, social structures, project management, quality, and support tools. These can be reconciled within the process domain along with concerns about metrics and standards, organizational procedures and norms and, if possible, all underpinned by a metamodel and ontology (e.g., Henderson-Sellers, 1995; Rolland & Prakash, 1996).

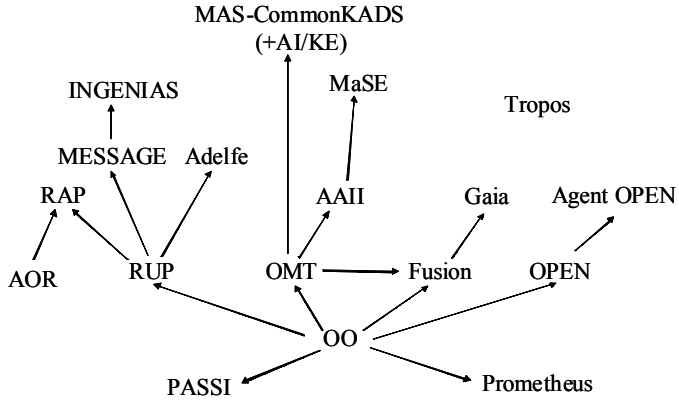
Any methodology also needs to contain sufficient abstractions to fully model and support agents and MASs—arguably, simple extensions of OO methodologies are too highly constrained by the sole focus on objects. Thus, an AO methodology needs to focus on an organized society of agents playing roles within an environment. Within such an MAS, agents interact according to protocols determined by the agents' roles.

We should also ask what it means for a methodology to be "agent-oriented" in the sense that we talk of an OO methodology in the context of object technology. In this case, however, object technology has two foci. In an OO methodology, we use OO concepts to describe the methodology, which, in turn, can be used to build object-oriented systems. In contrast, when we speak of an AO methodology, we generally do not mean a methodology that is itself constructed on agent-oriented principles but merely one that is oriented towards the creation of agent-based software. Thus, all the chapters except one follow this "definition." Only Tropos (described in detail in Chapter 2) claims to use "agent think" in its very derivation.

Genealogy of Methodologies

Agent-oriented methodologies have several roots. Some are based on ideas from artificial intelligence (AI), others as direct extensions of existing OO methodologies, whilst yet others try to merge the two approaches by taking a more purist approach yet allowing OO ideas when these seem to be sufficient. Figure 1

Figure 1. Direct and indirect influences of object-oriented methodologies on agent-oriented methodologies



shows these lineages and influences in what might be called a genealogy of the ten AO methodologies discussed in this book.

Several methodologies acknowledge a direct descendancy from full OO methods. In particular, MaSE (DeLoach, 1999; Wood & DeLoach, 2000) acknowledges influences from Kendall, Malkoun and Jiang (1996), as well as an heredity from AAII (Kinny, Georgeff, & Rao, 1996), which in turns was strongly influenced by the OO methodology of Rumbaugh and colleagues called OMT (Rumbaugh, Blaha, Premerlani, Eddy, & Lorensen, 1991). Similarly, the OO methodology of Fusion (Coleman, Arnold, Bodoff, Dollin, & Gilchrist, 1994) was said to be highly influential in the design of Gaia (Wooldridge, Jennings, & Kinny, 2000; Zambonelli, Jennings, & Wooldridge, 2003). Two other OO approaches have also been used as the basis for AO extensions. RUP (Kruchten, 1999) has formed the basis for ADELFE (Bernon, Gleizes, Picard, & Glize, 2002) and also for MESSAGE (Caire et al., 2001), which, in turn, is the basis for INGENIAS (Pavon, Gomez-Sanz, & Fuentes, 2005). More recently, RUP has also been used as one of the inputs (together with AOR [Wagner, 2003]) for RAP (Taveter & Wagner, 2005). Secondly, the OPEN approach to OO software development has been extended significantly to support agents, sometimes called Agent OPEN (Debenham & Henderson-Sellers, 2003). Finally, two other methodologies exhibit influences from object-oriented methodological approaches. Prometheus (e.g., Padgham & Winikoff, 2002a,b), although not an OO descendant, does suggest using OO diagrams and concepts whenever they exist and are compatible with the agent-oriented paradigm. Similarly, PASSI (2005) merges OO^3 and MAS ideas, using UML as its main notation.

Somewhat different is the MAS-CommonKADS methodology (Iglesias, Garijo, Gonzalez, & Velasco, 1996, 1998). This is the only solidly-AI-based methodology discussed in this book, yet it also claims to have been strongly influenced by OO methodologies, notably OMT.

Then there are the methodologies that do not acknowledge any direct genealogical link to other approaches, OO or AO. Discussed in this book is Tropos (Bresciani, Giorgini, Giunchiglia, Mylopoulos, & Perini, 2004; Castro, Kolp, & Mylopoulos, 2002; Giorgini, Kolp, Mylopoulos, & Pistore, 2004). Tropos has a significant input from *i** (Yu, 1995) and a distinct strength in early requirements modelling, focusing as it does on describing the goals of stakeholders that describe the "why" as well as the more standard support for "what" and "how." This use in Tropos of the *i** modelling language (particularly in the analysis and design phases) gives it a different look and feel from those that use Agent UML (a.k.a. AUML; Odell et al., 2000) as a notation. It also means that the non-OO mindset permits users of Tropos to take a unique approach to the modelling of agents in the methodological context. Other approaches not covered in this book include Nemo (Huget, 2002), MASSIVE (Lind, 1999), Cassiopeia (Collinot & Drogoul, 1998; Collinot, Drogoul, & Banhamou, 1996) and CAMLE (Shan & Zhu, 2004)—although in CAMLE there are some parallels drawn between its notion of "caste" and the concept of an OO class, as well as some connection to UML's composition and aggregation relationships.

Further comparisons of these methodologies are undertaken in Chapter 12, which complements and extends earlier framework-based evaluative studies of, for instance, Cernuzzi and Rossi (2002), Dam and Winikoff (2004), Sturm and Shehory (2004) and Tran, Low and Williams (2004).

Common Terms/Concepts Used in the AO Methodologies

The basic set of concepts underlying agent-oriented (AO) methodologies and the associated agent terminology are not universally agreed upon. Nevertheless, there is sufficient agreement to make it worthwhile for us to summarize commonly agreed upon terms here in Chapter 1 in order that authors of later chapters need not repeat this material. Bear in mind, however, that each methodological approach may treat these terms and their underpinning conceptual base slightly differently—as will be pointed out when necessary.

Agents are often contrasted with objects and the question —"What makes an agent an agent and not an object?" – is particularly difficult to answer when one considers the concepts of "active objects" in an OO modelling language such as the UML.

The novelty of agents is said to be that they are proactive (as well as reactive), have a high degree of autonomy, and are situated in and interact with their

environment (Zambonelli, Jennings, & Wooldridge, 2001), which is sometimes considered simply as a *resource*. This introduces (beyond objects) issues to do with not only the environment itself but also the direct interactions between the agent(s) and their environment. Thus, interfaces are as or more important in agents than in objects and, particularly, object-implemented components. When an agent perceives its environment, perhaps by means of a *sensor* of some kind, it is common to model that in terms of *percepts*.[4] When an agent interacts with its environment in order to make a change in the environment, it is called an *action*. The mechanism by which this is accomplished is often called an *effector* (see Chapter VI for further details).

Agent behaviour can be classified as *reactive* or *proactive*. A reactive agent only responds to its environment. These changes are communicated to the agent as *events* (although events also occur as a direct result of messages sent from other agents or indeed sent internally to the agent). Thus, changes in the environment have an immediate effect on the agent. The agent merely reacts to changing conditions and has no long-term objectives of itself. In contrast, a proactive agent has its own objectives. These are usually represented as one or more *goals* (e.g., Dardenne, Lamsweerde, & Fickas, 1993; Giorgini et al., 2004; Kendall & Zhao, 1998). To achieve a goal, it is usual to construct a *plan* and then to execute that plan by means of process elements known as *actions* (or often as *tasks*) (Figure 2). In reality, many agents are designed as hybrid agents, possessing both reactive and proactive characteristics. The challenge then is for the designer to balance these two very different behaviours in order to create an overall optimal behaviour.

One well-known agent architecture[5] that reflects many of these notions is the Beliefs, Desires, and Intentions (BDI) architecture of Rao and Georgeff (1995). Winikoff, Padgham, and Harland (2001) summarize this architecture in terms of three "abstraction layers" called philosophical (renamed here as psychological), theoretical, and implementation (Table 1). Beliefs, Desires, and Intentions are seen as high-level, abstract, externally ascribed characteristics. These three characteristics are then mapped through to the design or model layer. In particular, we note in Table 1 that Beliefs represent the agent's knowledge at various granularity levels, while Desires, which represent heterogeneous objectives possibly including some conflicts, map to Goals (now a consistent set of objectives) within the agent, and Intentions are mapped to Committed Goals (a coherent subset of goals with no conflicts or contradictions). In this table (from Henderson-Sellers, Tran, & Debenham, 2005), Plans are included specifically to account for the "how" element not originally formalized in the original BDI descriptions. Typically, each goal would have a link to at least one plan. These ideas have been described by Henderson-Sellers et al., 2005) by a metamodel fragment as shown in Figure 3.

Figure 2. Metamodel fragment showing, inter alia, the links between goals and tasks

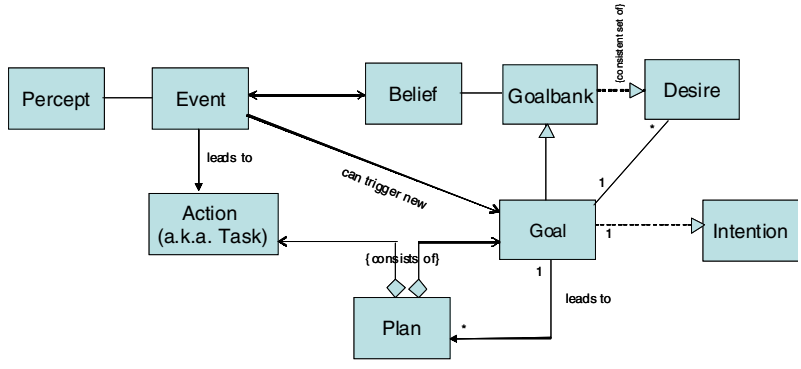
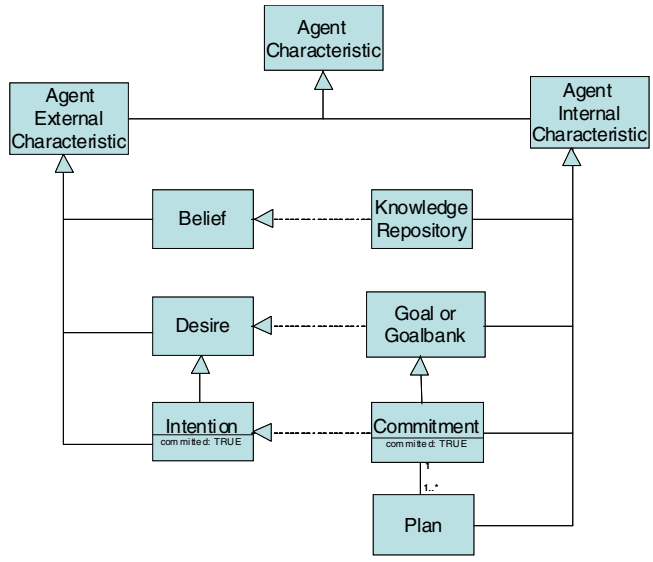


Table 1. Relationships between terminology of BDI model (after Henderson-Sellers et al., 2005)

Viewpoint		Column 3	Column 3 + Commitment	
Psychology	Belief	Desire	Intention	Wherewithal ("how")
Design/Model	World Model	Goal	Commitment	Plan
Implementation	Knowledge Base	-	-	Running (or instantiated) Plan

The BDI and similar models offer a description of the *internal* structure of an agent. However, agents are social entities and thus optimal performance is more likely from a cluster of agents. This is an MAS or multi-agent system. The methodologies in this book are aimed at building MASs and not single-agent systems. A metaphor that is often used is that of the (human) organization, in which the participating agents are said to exhibit *social* behaviour. Theories developed for human organizations are often usefully applied, such as that of social commitment (Cavedon & Sonenberg, 1998; Yolum & Singh, 2002) and social norms (Castelfranchi, Dignum, Jonker, & Treur, 2000). Within this social environment, as with humans, agents are seen to form teams (Cohen & Levesque, 1991) in which agents play roles (Kendall, 2000). In fact, the notion of a role, while supported in some OO approaches (e.g., Reenskaug, Wold & Lehne, 1996; Firesmith & Henderson-Sellers, 2002), is a major differentiating factor for agents. In fact, in the comparative surveys discussed above and further in Chapter XII, one common, major classification axis is whether the AO methodology is a role-based one or not.

Figure 3. Metamodel of concepts used in the BDI architecture (after Henderson-Sellers et al., 2005)



If agents are operating in a communal workplace, then clearly they need to communicate even more urgently than do objects. While objects react to incoming messages and also to events in most modern OO language, agents have the ability through their autonomy and proactive behaviour to cooperate and coordinate with each other by the exchange of messages. The major difference with objects is that agents can receive messages that are not confined to execution requests but can also consist of information or requests for information (Faber, 1999). Agent-to-agent communication is a major research area in MAS behaviour. We note here that the key issues are how to describe agent messages in terms of *interaction* and *communication* protocols, perhaps using a formal, mathematically based language.

An MAS clearly contains many agents within the contextual environment. In addition to inter-agent communication, we need to recognize that, within an MAS, agents need to both compete and cooperate. Although essentially selfish in their autonomy, agents act like humans: sometimes aiming to fulfil their own goals at the expense of all other agents/humans but mostly in a more social structure in which it is recognized that collaboration and sharing of work is mutually beneficial as well as individualistically profitable. Thus, the notion of agents organized to work within a social structure is also a very strong driver in AO methodologies (e.g., Zambonelli et al., 2001). Indeed, some of the methodologies discussed in this book argue that their main differentiator is that they

address these issues of social structure as a top priority, perhaps downplaying the discussions about agent infrastructure and whether models such as BDI are optimum.

Moving from agent infrastructure and communication to process elements, we should briefly outline the use of various terms such as lifecycle, analysis, design, and implementation. In the 1970s and 1980s, these terms were generally understood and related to organizational structure within software development companies, particularly those using the waterfall approach. This described a number of "phases," the totality of which was called the software development life cycle or SDLC. However, the advent of object technology and OO software development methodologies throughout the 1990s led to an avoidance of these terms on the macro-scale.

Analysis is equated to the understanding of something already in existence, sometimes labelled "discovery." It relates to the problem space. In contrast, design is considered as part of the solution space in which various possible "answers" are considered; thus, it could be labelled "invention" (Booch, 1994). The argument is that these do not occur sequentially on a timescale of months but in normal human cognition on a timescale of seconds and highly iteratively. Together, these two obsolescent phases were frequently called "modelling" in OO methodologies. This leads to a modelling language such as UML potentially having an equivalent scope, that is, "analysis" and "design" (although, in reality, UML is heavily biased towards "design"). This, in turn, often leads to confusion since clearly the "modelling phase" is a lengthy one in which initially there is more analysis than design, whereas towards the end of the phase there is more design than analysis going on. The use of any particular technique or modelling notation thus shifts in balance from those more useful for "discovery" to those more focussed on the solution space. Notwithstanding, it is sometimes useful to re-introduce the analysis and design terms, not as straitjackets for two lengthy sequential, non-iterative phases, but simply to remind the reader and user of the methodology of the shifting balance described above. This then permits discussion of more "analysis-type/design-type" techniques under the banner of an "analysis/design phase."

Since different methodology authors have different models at a granularity beneath the overall SDLC, we will not attempt here to prescribe the internal structure of the SDLC but will leave that to individual chapter authors. An emerging framework that is hinted at in some chapters is Model-Driven Architecture (MDA). This is a fairly recent initiative of the Object Management Group (OMG) that attempts to supply an architectural infrastructure to SDLC by identifying a Platform-Independent Model (PIM) that avoids any assumptions about operating system, programming language, hardware, and so forth. This PIM is then translated (the aim is to eventually be able to do this automatically) to a Platform Specific Model or PSM (Kleppe, Warmer, & Bast, 2003). The

applicability of this to AO methodologies has not yet been fully explored – initial steps towards this goal are discussed by Taveter and Wagner (2005).

Introducing the Methodologies

The methodologies selected for this volume straddle object-oriented and AI concepts. We begin with a methodology (Tropos) that avows to use the agent paradigm in its very design, as well as being appropriate, as all the others in this book, for the development of software systems aligned with the agent-oriented paradigm of computing. The next two methodologies to be considered, MAS-CommonKADS and PASSI, both epitomize the bridging to be undertaken between OO and AI, while in Prometheus, OO concepts are used when they are applicable but are otherwise eschewed. Gaia is a methodology that has been highly influenced by object technology yet retains a truly agent-oriented feel.

The next group of four methodologies centers on specific extensions to the object-oriented methodology, RUP. The specific agent extensions portrayed here are ADELFE, which specializes in adaptive agents, MESSAGE (and its "offspring" INGENIAS) together with RAP, which utilizes not only RUP but also the modelling language of AOR, created by the same authors. A second OO influence, that of the older approach of OMT, is seen in our last methodology, MaSE, which is influenced also by the AAII work of a decade ago and the influential role modelling work of Kendall and colleagues (Kendall, Malkoun, & Jiang, 1996).

Acknowledgments

This is Contribution number 04/30 of the Centre for Object Technology Applications and Research.

References

AOS (2000). *JACK Intelligent Agents User Guide,* AOS Technical Report, Agent Oriented Software Pty Ltd, July. Retrieved from: *http://www.jackagents.com/docs/jack/html/index.html*

Bernon, C., Gleizes, M.-P., Picard, G., & Glize, P. (2002). The ADELFE methodology for an intranet system design. In P. Giorgini, Y. Lespérance, G. Wagner, & E. Yu (Eds.), *Proceedings of Agent-Oriented Information Systems, AOIS-2002* (p. 1-15). AOIS.org.

Booch, G. (1994). *Object-oriented analysis and design* (2nd ed.). Redwood City, CA: The Benjamin/Cummings Publishing Company, Inc.

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopolous, J., & Perini, A. (2004). Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems, 8*(3), 203-236.

Burrafato, P. & Cossentino, M. (2002). Designing a multi-agent solution for a bookstore with the PASSI methodology. In P. Giorgini, Y. Lespérance, G. Wagner & E. Yu (Eds.), *Proceedings of the Agent-Oriented Information Systems 2002* (pp. 102-118). AOIS.org.

Caire, G., Coulier, W., Garijo, F., Gomez, J., Pavon, J., Leal, F., Chainho, P., Kearney, P., Stark, J., Evans, R., & Massonet, P. (2001). Agent-oriented analysis using MESSAGE/UML. In M. Wooldridge, G. Wei, & P. Ciancarini (Eds.), *Agent-oriented software engineering II* (p. 119-135). LNCS 2222. Berlin: Springer-Verlag.

Castelfranchi, C., Dignum, F., Jonker, C., & Treur, J. (2000). Deliberate normative agents: Principles and architectures. In N. Jennings & Y. Lespérance (Eds.), *Intelligent agents VI* (p. 364-378). Berlin: Springer-Verlag.

Castro, J., Kolp, M., & Mylopoulos, J. (2002). Towards requirements-driven information systems engineering: The Tropos project. *Information Systems*, *27*(6), 365-389.

Cavedon, L. & Sonenberg, L. (1998). On social commitment, roles and preferred goals. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS),* July 3-7, Paris (pp. 80-87). IEEE Computer Society.

Cernuzzi, L. & Rossi, G. (2002). On the evaluation of agent oriented methodologies. In *Proceedings of OOPSLA 2002 Workshop on Agent-Oriented Methodologies* (pp. 21-30). Sydney, AUS: Centre for Object Technology Applications and Research.

Chan, K., Sterling, L., & Karunasekera, S. (2004). Agent-oriented software analysis. In *Proceedings of 2004 Australian Software Engineering Conference* (pp. 20-27). Los Alamitos, CA: IEEE Computer Society Press.

Cohen, P.R. & Levesque, H.J. (1991). Teamwork. *Nous*, *25*(4), 487-512.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., & Gilchrist, H. (1994). *Object-oriented development. The fusion method.* Englewood Cliffs, NJ: Prentice Hall.

Collinot, A. & Drogoul, A. (1998). Using the Cassiopeia method to design a soccer robot team. *Applied Artificial Intelligence (AAI) Journal*, *12*(2-3), 127-147.

Collinot, A., Drogoul, A., & Benhamou, P. (1996). Agent-oriented design of a soccer robot team. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS'96)* (pp 41-57). Menlo Park, CA: American Association for Artificial Intelligence.

Cossentino, M. (2005). From requirements to code with the PASSI methodology. In B. Henderson-Sellers & P. Giorgini (Eds.), *Agent-oriented methodologies* (Chapter 4). Hershey, PA: Idea Group.

Cossentino, M. & Potts, C. (2002). A CASE tool supported methodology for the design of multi-agent systems. In H.R. Ababnia & Y. Mun (Eds.), *Proceedings of the 2002 International Conference on Software Engineering Research and Practice (SERP'02),* Las Vegas, June 24-27 (pp. 315-321).

Dam, K.H. & Winikoff, M. (2004). Comparing agent-oriented methodologies. In P. Giorgini, B. Henderson-Sellers, & M. Winikoff (Eds.), *Agent-oriented systems* (pp. 78-93). LNAI 3030. Berlin: Springer-Verlag

Dardenne, A., Lamsweerde, A. v., & Fickas, S. (1993).Goal-directed requirements acquisition. *Science of Computer Programming, 20,* 3-50.

Debenham, J. & Henderson-Sellers, B. (2003). Designing agent-based process systems - Extending the OPEN Process Framework. In V. Plekhanova (Ed.), *Intelligent agent software engineering* (Chapter VIII, pp. 160-190). Hershey, PA: Idea Group Publishing.

DeLoach, S.A. (1999). Multiagent systems engineering: A methodology and language for designing agent systems. In *Proceedings of the First International Bi-conference Workshop on Agent-Oriented Information Systems (AOIS '99),* May 1, Seattle. AOIS.org.

Faber J. (1999). *Multi-agent systems: An introduction to distributed artificial intelligence*. Reading, MA: Addison-Wesley.

Firesmith, D.G. & Henderson-Sellers, B. (2002). *The OPEN process framework.* Harlow, UK: Addison Wesley.

Giorgini, P., Kolp, M., Mylopoulos, J., & Pistore, M. (2004). The Tropos methodology: An overview. In F. Bergenti, M.P. Gleizes, & F. Zambonelli (Eds.), *Methodologies and software engineering for agent systems* (pp. 89-106). Boston: Kluwer Academic Publishing.

Henderson-Sellers, B. (1995). Who needs an OO methodology anyway? *Journal of Object Oriented Programming*, *8*(6), 6-8.

Henderson-Sellers, B., Tran, Q.-N.N., & Debenham, J. (2005). An etymological and metamodel-based evaluation of the terms "goals and tasks" in agent-oriented methodologies. *J. Object Technol., 4*(2), 131-150.

Henderson-Sellers, B. & Unhelkar, B. (2000). *OPEN modeling with UML*. London: Addison-Wesley.

Huget, M.-P. (2002). Nemo: An agent-oriented software engineering methodology. In *Proceedings of OOPSLA 2002 Workshop on Agent-Oriented Methodologies* (pp. 43-53). Sydney, AUS: Centre for Object Technology Applications and Research.

Iglesias, C.A., Garijo, M., Gonzalez, J.C., & Velasco, J.R. (1996). A methodological proposal for multiagent systems development extending CommonKADS. In *Proceedings of 10th KAW*, Banff, Canada. Available online *http://ksi.cpsc.ucalgary.ca/KAW/KAW96/KAW96Proc.html*

Iglesias, C.A., Garijo, M., Gonzalez, J.C., & Velasco, J.R. (1998). Analysis and design of multi-agent systems using MAS-CommonKADS. In M.P. Singh, A. Rao, & M.J. Wooldridge (Eds.), *Intelligent agents IV: Agent theories, architectures, and languages* (LNAI Vol. 1365, pp. 313-326). Berlin: Springer-Verlag.

Iivari, J., Hirschheim, R., & Klein, H.K. (1999). Beyond methodologies: Keeping up with information systems development approaches through dynamic classification. In *Proceedings of HICSS 1999* (p. 7044). Los Alamitos, CA: IEEE Computer Society Press.

Jayaratna, N. (1994). *Understanding and evaluating methodologies, NISAD: A systematic framework.* Maidenhead, UK: McGraw-Hill.

Kendall, E.A. (2000). Software engineering with role modelling. In *Proceedings of the Agent-Oriented Software Engineering Workshop* (pp. 163-169). LNCS, Vol. 1957. Berlin: Springer-Verlag.

Kendall, E.A., Malkoun, M.T., & Jiang, C. (1996). A methodology for developing agent based systems for enterprise integration. In P. Bernus & L. Nemes (Eds.), *Modelling and methodologies for enterprise integration.* London: Chapman and Hall.

Kendall, E.A. & Zhao, L. (1998). Capturing and structuring goals. Presented at the *Workshop on Use Case Patterns, Object Oriented Programming Systems Languages and Architectures*, Vancouver, BC, Canada, October 18-22.

Kinny, D., Georgeff, M., & Rao, A. (1996). A methodology and modelling techniques for systems of BDI agents. Technical Note 58, Australian Artificial Intelligence Institute, also published in *Proceedings of Agents Breaking Away, the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96)* (pp. 56-71). Springer-Verlag.

Kleppe, A., Warmer, J., & Bast, W. (2003). *MDA explained: The model driven architecture - Practice and promise.* Reading, MA: Addison-Wesley.

Kruchten, P. (1999). *The rational unified process. An introduction*. Reading, MA: Addison-Wesley.

Lind, J. (1999). *Iterative software engineering for multiagent systems. The MASSIVE Method*. LNAI 1994. Berlin: Springer-Verlag.

Luck, M., Ashri, R., & D'Inverno, M. (2004). *Agent-based software development*. Boston: Artech House.

Martin, J. & Odell, J.J. (1995). *Object-oriented methods: Pragmatics and considerations*. Upper Saddle River, NJ: Prentice-Hall.

Milgrom, E., Chainho, P., Deville, Y., Evans, R., Kearney, P., & Massonet, P. (2001). *MESSAGE: Methodology for engineering systems of software agents. Final guidelines for the identification of relevant problem areas where agent technology is appropriate*. EUROSCOM Project Report P907. Available online *http://www.eurescom.dr/~publicwebspace/P800-series/P815/web/index.htm*

Odell, J., Van Dyke Parunak, H., & Bauer, B. (2000). Extending UML for agents. In G. Wagner, Y. Lesperance & E. Yu (Eds.), *Proceedings of Agent-Oriented Information Systems Workshop* (pp. 3-17). *17th National Conference on Artificial Intelligence*, Austin, TX.

OMG (2001). OMG Unified Modeling Language Specification, Version 1.4. September 2001. OMG document formal/01-09-68 through 80 (13 documents). Available online *http://www.omg.org*

Padgham, L. & Winikoff, M. (2002a). Prometheus: A methodology for developing intelligent agents. In F. Giunchiglia, J. Odell, & G. Weiß (Eds.), *Agent-oriented Software Engineering III Proceedings of the Third International Workshop on Agent-Oriented Software Engineering (AAMAS'02)* (pp. 174-185). LNCS 2585.

Padgham, L. & Winikoff, M. (2002b). Prometheus: A pragmatic methodology for engineering intelligent agents. In J. Debenham, B. Henderson-Sellers, N. Jennings, & J.J. Odell (Eds.), *Agent-oriented Software Engineering III Proceedings of the Workshop on Agent-oriented Methodologies at OOPSLA 2002*, November 4, Seattle (pp. 97-108). Sydney: Centre for Object Technology Applications and Research.

Pavón, J., Gomez-Sanz, J., & Fuentes, R. (2005). The INGENIAS methodology and tools. In B. Henderson-Sellers & P. Giorgini (Eds.), *Agent-oriented methodologies* (Chapter 4). Hershey, PA: Idea Group.

Rao, A.S. & Georgeff, M.P. (1995). BDI agents: From theory to practice. In V.R. Lesser & L. Gasser (Eds.), *Proceedings of the First International*

Conference on Multi-Agent Systems, San Francisco (pp. 312-319). Cambrige, MA: MIT Press.

Rolland, C. & Prakash, N. (1996). A proposal for context-specific method engineering. In *Proceedings of the IFIP WG8.1 Conference on Method Engineering* (pp. 191-208). London: Chapman and Hall.

Rolland, C., Prakash, N., & Benjamen, A. (1999). A multi-model view of process modelling. *Requirements Eng. J.*, *4*(4), 169-187.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W. (1991). *Object-oriented modeling and design*. Englewood Cliffs, NJ: Prentice-Hall.

Shan, L. & Zhu, H. (2004). Software engineering for multi-agent systems III: Research issues and practical applications. In R. Choren, A. Garcia, C. Lucena, & A. Romanovsky (Eds.), *Proceedings of the Third International Workshop on Software Engineering for Large-Scale Multi-Agent Systems,* (pp. 144-161). Berlin: Springer-Verlag.

Shoham, Y. (1993). Agent-oriented programming. *Artificial Intelligence*, *60*(1), 51-568.

Sturm, A. & Shehory, O. (2004). A framework for evaluating agent-oriented methodologies. In P. Giorgini, B. Henderson-Sellers, & M. Winikoff (Eds.), *Agent-oriented systems* (pp. 94-109). LNAI 3030. Berlin: Springer-Verlag.

Taveter, K. & Wagner, G. (2005). Towards radical agent-oriented software engineering processes based on AOR modelling. In B. Henderson-Sellers & P. Giorgini (Eds.), *Agent-oriented methodologies* (Chapter 10). Hershey, PA: Idea Group.

Tran, Q.-N.N., Low, G., & Williams, M.-A. (2004). A preliminary comparative feature analysis of multi-agent systems development methodologies. In *Proceedings of AOIS@CAiSE*04*, Faculty of Computer Science and Information, Riga Technical University, Latvia (pp. 386-398).

Wagner, G. (2003). The agent-object relationship metamodel: Towards a unified view of state and behaviour. *Inf. Systems, 28*(5), 475-504.

Winikoff, M., Padgham, L., & Harland, J. (2001). Simplifying the development of intelligent agents. In M. Stumptner, D. Corbett, & M. J. Brooks (Eds.), *Proceedings of the 14th Australian Joint Conference on Artificial Intelligence (AI'01),* Adelaide, 10-14 December (pp. 557-558). LNAI 2256, Springer-Verlag.

Wood, M. & DeLoach, S.A. (2000). An overview of the multiagent systems engineering methodology. In P. Ciancarini & M. Wooldridge (Eds.), *Proceedings of the 1st International Workshop on Agent-Oriented*

Software Engineering (AOSE-2000) (pp. 207-222). LNCS 1957, Springer-Verlag.

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

Yolum, P. & Singh, M.P. (2002). Flexible protocol specification and execution: Applying event calculus planning using commitments. In *Proceedings of the 1st Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)* (pp. 527-534). New York: ACM Press.

Yu, E. (1995). *Modelling strategic relationships for process reengineering*. PhD Thesis, University of Toronto, Department of Computer Science.

Zambonelli, F., Jennings, N. R., & Wooldridge, M. (2001). Organisational abstractions for the analysis and design of multi-agent systems. In *Proceedings of the Agent-Oriented Software Engineering Workshop* (pp. 235-251). LNCS, Vol. 1957. Berlin: Springer-Verlag.

Zambonelli, F., Jennings, N., & Wooldridge, M. (2003). Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology*, *12*(3), 317-370.

Endnotes

1 It has also been argued (Chan, Sterling, & Karunasekera, 2004) that use of AO analysis may be beneficial even if the implementation is not in an AO language but, say, uses object-oriented design and programming.

2 If not, a useful introduction is to be found in Henderson-Sellers and Unhelkar (2000)

3 In earlier publications (e.g., Burrafato & Cossentino, 2002; Cossentino & Potts, 2002) this was not obvious.

4 Since agents are part of the environment, messages received from other agents can also be considered as percepts.

5 See also discussion in Chapter 10.