Chapter 3

A SURVEY ON AGENT-ORIENTED ORIENTED SOFTWARE ENGINEERING RESEARCH

Jorge J. Gomez-Sanz
Facultad de Informatica,
Universidad Complutense de Madrid, 28040 Madrid, Spain
jjgomez@sip.ucm.es

Marie-Pierre Gervais
Laboratoire d'Informatique de Paris 6
8 rue du Capitaine Scott, 75015 Paris, France
Marie-Pierre.Gervais@lip6.fr

Gerhard Weiß
Institut für Informatik,
TUM Boltzmannstrasse 3, 85748 Garching, GERMANY
weissg@informatik.tu-muenchen.de

Abstract This chapter presents a selection of current research works on agent technology which are focused on the development of Multi-Agent Systems. The purpose of this chapter is to guide developers through existing theories, methods, and software that can be applied in each stage of a development. However, this guide is not exhaustive due the amount of agent-related research works. Thus authors have added references to consult other information sources and complement the information given here. Readers are encouraged to consult these external references in order to obtain a more accurate view of the field.

1. Introduction

This past decade, developing a Multi-Agent System (MAS) has evolved from an art to a structured discipline. Existing results in MAS research enable a developer to construct MAS easier than before. Among others, there are tools that can produce complete MAS from a specification, libraries of components that deal with concrete MAS issues (distributed planning, reasoning, learning), and theories that describe MAS behavior and properties. Knowing all of them requires a great effort. Existing surveys facilitate this task, but it is hard to give an overall view of what software, theories, and methodologies exist, and how they are applied to MAS development.

To alleviate this problem, and make the information easier to apprehend, authors of this chapter have structured existing references into sections that deal with MAS development from an engineering point of view. Thus, there are sections that consider *analysis*, *design*, *implementation*, and *testing*. The purpose is to make this chapter less a survey and more a manual for MAS development. This way, developers with some background on conventional Software Engineering (SE) will see how they would do with agents what they do using other paradigms. Also, this approach benefits beginners and specialized researchers in this field by introducing a general picture of the development, not just solutions to concrete problems.

The criteria to distribute agent research into these stages have been obtained from the SE itself. In SE, each stage of a development process deals with concrete topic and pursues the production of certain documentation, specification or software. Trying to adapt topics considered in MAS research and the topics associated to each development stage, the authors of this paper have prepared a brief summary of what readers can find along the chapter:

- Analysis. This section deals with research work in agents helping to obtain a problem description. In traditional software engineering, this relates to the expression of *requirements*. In the agent domain, there are agent approaches that deal directly with these *requirements*, but there are also other hybrid approaches that have been adapted to the agent concepts. This section includes considerations about the role of agents in analysis purposes, specification languages that use agent concepts, and support tools for these languages.

- Design. This section considers how to facilitate a design of a MAS using agent concepts and technology. In software engineering, de-

sign covers the study of how to realize analysis elements into another specification - software architecture, components, expected behaviors - that can be directly implemented. To translate analysis specifications, it is necessary to know how to build agents from scratch and using agent development environments.

- Implementation. This section surveys different approaches to use agent technology in realizing concrete agent oriented designs. It proposes several agent oriented languages, software libraries, frameworks, and support tools.

- Testing. This section is a review of testing methods based on agent concepts enabling to check that a MAS satisfies initial requirements and that it has been built with no errors.

It should be remarked that the results mentioned in these sections are focused in the Agent Oriented Software Engineering (AOSE) domain. There is no intention to evaluate state of art of Software Engineering (SE). In principle, existing formalisms, techniques and methods (e.g., object-oriented ones) can be applied to develop agent-oriented software. However, a problem with these approaches is that they fail, or tend to fail to capture the essence of agent orientation [Jennings, 2000]. Consequently, approaches devoted to the agent orientation are needed and such approaches are surveyed in this chapter. For the sake of accuracy, the chapter uses software engineering terminology when possible. This terminology is found in [Sommerville, 2001] and [Pressman, 1982].

Each section contains recommendations of the authors of this survey about specialized literature and software. Though authors of this chapter have tried not to forget any relevant research, reading this chapter is not sufficient to obtain an adequate knowledge of the area. So, it is strongly recommended to read the papers referred in this work, as well as subscribing to organizations like AgentLink or AgentCities in order to be up to date with new results from research work.

2. Analysis

The analysis bases on the obtention of requirements in order to derive a description of what has to be built. A requirement determines what a software system should do and defines constraints on its operations and implementation. According to [Sommerville, 2001], there are several types of requirements. Among others, there are user requirements, system requirements, software, functional, and non-functional requirements. The process for obtaining requirements is not trivial. In fact,

there is a discipline, named *Requirements Engineering* (RE), with this purpose. RE will be reviewed later in this section.

Agent research centers in functional and non-functional requirement definition. Functional requirements are statements of the services that the system must provide, or descriptions of how some computations must be carried out. Non-functional requirements are product requirements which constrain the system being developed, process requirements which apply to the development process and external requirements [Sommerville, 2001]

Despite the type of requirement, gathering them does not imply the choice of an agent model, since most of existing approaches make general assumptions about agents. A model of agent determines what elements are required to specify agents and establishes a predefined behavior. A comprehensive collection of articles, and a survey of features of existent agent models, can be found in [Huhns and Singh, 1997]. Choosing an agent model at this point may couple too early a development with different agent architectures. So it may be better, unless it is a requirement, to leave this choice to the design stage.

Requirements themselves are understood in different ways by researchers in agents. Existing results can be categorized in one of these:

- Agent approaches oriented towards requirements representation. These representations include the concept *requirement* as first class citizen. These works are referred in the RE literature as well.
- Agent approaches that represent the system to be built using formal methods. These methods reuse formalisms tested in SE. They are directed towards the obtention of a formal specification.
- Agent approaches that represent the system to be built using diagrams. A frequent solution in SE is the use of diagrams to represent different aspects of a system. Agents have followed this line proposing different types of diagrams and concepts to capture the internals and externals of MAS. The specification obtained does not need to be formal.

Developers can decide at this moment what kind of analysis is more adequate to their situation. To present research works in each category, the chapter have been divided into three subsections.

2.1 Agent approaches specialized in defining requirements

This section studies the role of agents in RE. Applying agents in RE implies what some authors name *Agent Oriented Requirement Engineer-*

ing (AORE). These approaches are characterized by ascribing a more important role to the *agents*. However, according to surveys like [van Lamsweerde, 2000], the *goal* concept is more extended than *agent* concept to represent requirements. As this may disorient non-experienced developers, it is suggested that they read [Yu and Mylopoulos, 1998], which includes an extensive argumentation in favor of using *goal* as a first class concept when determining requirements. To have a pure, i.e. a non agent oriented view of requirements engineering, readers can consult [Zave, 1997] or [Nuseibeh and Easterbrook, 2000]. The main difference of these works with respect to [van Lamsweerde, 2000] is that the *agent* concept is not so important. [Zave, 1997] categorizes requirements engineering works taking into account the kinds of problems that are addressed, and types of solutions to these problems. [Nuseibeh and Easterbrook, 2000] is a detailed review of RE that introduces different stages identified in RE processes and related works for each one.

Integrating agents and RE is not easy. For instance, [Yu, 1997b] indicates that *agent* is a concept to be delineated with different requirements like *being distributed*, *being intelligent*, or *being autonomous*. The problem is that the *agent* concept also ties in with concrete implementations that already carry their requirements. So, to make sense an agent-oriented approach in RE, it is better to use initially more abstract concepts like *role* or *actor* and once what is required from an agent is clear, assign roles to agents. Later on, agents will be implemented so that initial requirements hold.

One of the main references in AORE is i* [Yu, 1997a]. It is a framework that uses actors, beliefs, commitments, and goals to model organizational environments and their information systems. There are examples of using i* in [Yu et al., 2001], that indicates how to identify the importance of a piece of information in a organization, and [Yu, 1999], that models a real furniture company. More examples can be found in the home page of the author [Yu, 03a]. i* has been also the starting point for other frameworks, concretely *Non-Functional Requirements* (NFR) [Chung et al., 2000] and *Goal-oriented Requirement Language* (GRL) [Yu and Liu, 2002]. There are examples and tutorials available for GRL at [Yu, 03b]. i* has a support tool called *Organization Modelling Environment* (OME) [Yu and Liu, 03]. This tool can be downloaded if previously a license agreement is sent to authors. OME also supports other notations, concretely GRL and NFR. So the same tool gives support for slightly different RE approaches.

Recently, i* has been adopted as underlying framework for an AOSE methodology named Tropos [Mylopoulos and Castro, 2000]. Tropos has added to i* a development process and automated translation methods

from a i* specification to agents based on Jack agent platform [Busetta et al., 1999].

Besides i*, there are also two other classic works like KAOS methodology [Dardenne et al., 1993] and Albert [Dubois et al., 1994]. KAOS also considers agents, actions, entities consumed by actions, and other relationships among entities. Agents are responsible for the execution of tasks in order to satisfy a goal, i.e., a requirement. An example of how to apply KAOS is [van Lamsweerde and Massonet, 1995], a paper that describes a distributed meeting scheduler. Notation of KAOS appears in [Dardenne et al., 1993]. KAOS has also its support tool, which is named GRAIL [Darimont et al., 1997].

Albert is a pure specification language with more detailed semantics than KAOS or i* semantics. In fact, it is less ambiguous than other approaches reviewed here. So its presentation has been moved to the next section.

Previous approaches lack of elements to model requirements with respect to agent interaction. In this sense, a recurrent solution is *role modeling*. It is oriented towards functional requirements, expressed with tasks, services, and roles integrated in workflows. Research from Kendall [Kendall, 1998] is a classic in this line of research. There are studies in identifying roles [Kendall, 1998] inside a development process though most of the work is done at the analysis level. This work has had a big influence on other works, such as ZEUS [Nwana et al., 1999] or MaSE [DeLoach, 2001]. Kendall's ideas constitute the method applied in the initial steps of ZEUS methodology [Collis and Ndumu, 1999] and MaSE [Wood and DeLoach, 2001]. For a more object-oriented approach, readers can consult [Depke et al., 2001], that considers functional requirements as UML use cases where roles participate in performing certain tasks. ODAC is also a methodology adapting standards from distributed object computing to agent and using UML [Gervais, 2003]. More details on approaches using UML are given in the *UML based approaches* section.

2.2 Analysis applying formal methods

Formal methods are mathematical modelling techniques applied to a software engineering process in order to obtain a non-ambiguous correct specification of the system to be built [Pressman, 1982]. The expected output of a formal method is a formal specification. This specification can be used with different purposes, besides formal verification, as it is mentioned in the tests section. As [Wooldridge, 1997] remarks, a formal

specification can be compiled to executable code or interpreted directly, without user intervention.

There are several research works that show how to apply formal methods in generating a specification of a MAS.

Abstract State Machines [Gurevich, 1984] are the formalism employed in MAS-CommonKADS [Iglesias et al., 1998a] to represent behavior of the system at a high level. Concretely, MAS-CommonKADS uses *Specification and Description Language* (SDL) [Union, 9 AD], a standard language used to describe systems in the telecommunications domain. However, the degree of details in SDL is quite high. This formalism is complemented with Class Responsibility Collaborator (CRC) cards as a representation method to gather information about different aspects of the system, like tasks specification or the purpose of a system service. CRC cards are templates with slots that developers must fill using a concrete language. Other works reuse state machines, like agenTool [DeLoach, 2001] that uses them to represent the behavior of internal components of agents and protocols as well. This solution shares with SDL a common view of communication as interchanged messages among different state machines. In this line of research, developers can consult [Rosenschein and Kaelbling, 1995], that show how to translate first order logic formulae to state machines using automated methods. Though it is not exactly the same, Petri Nets have been used many times to express the behavior of a component. As an example to its application to the MAS domain, readers can consult [Xu et al., 2002] and [Demazeau, 1995]. The first directly models the internal behavior of agents with this formalism. The second introduces Petri Nets as an abstraction to implement agent synchronization.

Z [Spivey, 1992b] is a mathematical formalism based on sets manipulation. There have been experiments in adapting Z to the agent domain [d'Inverno, M., Hindriks and Luck, 2000] [d'Inverno and Luck, 1996]. Benefits of using Z are mainly reusing the big amount of software and tools available to this language, what includes automated code generation and formal verification methods.

Logics appear frequently as formalism to represent agent behavior, perhaps due to the the extensive use of logics in classic artificial intelligence. For a review of logics in a MAS context, readers can consult [Singh, 1997]. A short introduction of modal logic to MAS appears in [van der Hoek, 2001] and [Wooldridge and Jennings, 1995a]. According to these reviews, modal logics are playing a main role in the agent research field. This is a novelty, since AI centers on propositional logic to represent knowledge and reasoning [Genesereth and Nilsson, 1987] with few attention to modal logic. In the agent domain, modal logic can han-

dle formulas that do not satisfy *extensionality*. *Extensionality* says that, to determine the truth-value of a formula, only the truth-value of its subformulas must be considered. Due this property, time and desire cannot be modelled in classic logic, see [van der Hoek, 2001] for further explanations. Main works in modal logics in this field are the formalization of BDI [Bratman, 1987] model in [Rao and Georgeff, 1991] and intentional logic from [Cohen and Levesque, 1990]. The first one was extended with architectures and methodologies [Kinny et al., 1996] and is referred in most works in the field. The second remarks the role of intentions using *intentional logics*. It centers on how beliefs, desires, intentions relate to agent actions. As a detailed example of a framework with BDI formalization and tool support, readers should review DESIRE [Brazier et al., 1997] [Brazier et al., 1994], framework for *DEsign and Specification of Interacting REasoning components*. Another relevant work in temporal logics, which are modal logics, is Concurrent-MetaMem [Fisher, 1994], though it relates to agent implementation rather than specification. However, it is interesting to read [Fisher, 1995a] since it sketches how to apply this logics in a software process as a whole. Though it does not focus in the whole process, Albert [Dubois et al., 1994] proposes a variant of temporal logic with extensions to consider actions, agents, and constraints of the behavior of the system. This language, mentioned in the previous section, was designed to gather requirements that later could be processed using formal tools, like theorem proving software. The web site of Albert is [Schobbens, 2003] to know more details or contact authors.

GAIA [Wooldridge et al., 2000] also applies logics to describe some aspects of the system. Like MAS-CommonKADS, uses set of CRC-like cards whose slots are filled with logic formulae. It is not completely formal because these logic formulae, in the available examples, are not tied to any model or implementation, so their interpretation depends on the developer. However, these can be considered as a good approach to integrate SE methods together with formal ones.

Situation calculus [McCarthy and Hayes, 1981] is another formalism applied to model MAS. Situation Calculus is a first-order language (with some second-order features) for representing dynamic domains. In the agent domain, a key work is ConGOLOG [Giacomo et al., 2000], a concurrent language to generate computational models that constraints task execution according to their preconditions and side-effects. The computational model that it defines executes tasks if and only if their execution will not take the system to instability. Though computationally it is an expensive problem, they have relaxed ConGOLOG constrains so that costs are affordable.

2.3 Analysis using diagrams

Some methodologies represent part of the analysis results using diagrams. Diagram based representations may not be formal. Some approaches propose diagrams that are interpreted by a developer, like UML. In those approaches, different developers can derive different interpretations. However, there are also diagrams that are not ambiguous at all, like Petri-Net graphic representations of a protocol or Entity-Relationship diagrams to represent databases tables. This section reviews two kinds of diagram based approaches. The first deals with UML applied to the agent domain. The second with meta-modelling languages as specification language of the MAS at the analysis level.

UML based approaches One the most widespread approach is AUML [Bauer et al., 2001], a project aiming at bringing agent concepts into UML [OMG, 2000c]. As a suggestion, readers should start by [Odell et al., 2000] since it is one of the first papers about AUML. One of the relevant results of AUML is protocol diagrams notation, which is being considered as a new notation for the standard UML to express concurrence and decision. [AUML Team, 03] is AUML web site. It contains working documents and articles about applying AUML to model different aspects of a MAS [Bauer et al., 2001] [Odell et al., 2001]. Unfortunately, there are no support tools for AUML.

A Process for Agents Societies Specification and Implementation (PASSI) [CSAI LAB, 03] is recent work that reuses UML tools as front-end. It applies a UML representation of elements belonging to an architecture for a better understanding and handling. Concretely, Rational Rose is supplemented with a customized plugin that provides PASSI extra functionality. As an example of PASSI modelling, readers can consult [Burrafato and Cossentino, 2002] that shows modelling of a book store company. In this line of research, something similar has been proposed in [Bergenti and Poggi, 2001]. It is a framework and a programming language that facilitates the definition of planning capabilities of agents. This approach inputs an XMI [OMG, 03] description of UML diagrams to generate code directly to a target agent platform. XMI is a model driven XML Integration framework for defining, interchanging, manipulating and integrating XML data and objects. As it is a standard (issued by OMG), any UML compliant tool should be valid as front-end.

Though these research works are very complete, they do not address properly MAS definition using a development process. Actually, they propose a too simple development process. Those readers interested in a more detailed solution to perform analysis, may read ADELFE [Carole

Bernon, Marie-Pierre Gleizes, Sylvain Peyruqueou, 2002]. It proposes a very detailed analysis method reusing UML notation that can be adapted to several agent models. The point with this work is that it offers a general view of what elements are implied in a MAS generation, which is complementary of [Wooldridge, 1997].

Meta-modeling as specification language. Meta-modelling is a technique for model description. Meta-modelling consists in describing types of objects, their properties, relationships and how they appear together in a model. This description is called a meta-model. A model is the instance of a meta-model and it conforms to a set of constraints defined in the meta-model, like *among objects of type A and B there can be only relationships one to one of type C*. Readers interested in knowing more about meta-modelling can consult [OMG, 2000b] or [OMG, 2000c], where a meta-modelling language is presented and applied to describe application data, program interfaces, or diagrams.

Why using meta-models in the agent domain? Citing [Gomez-Sanz et al., 2002], meta-models are useful as a kind of templates for generating agent models. They describe what any MAS should have. With meta-models of a MAS, the mission of the engineer turns into instantiating these meta-models in order to define the entities that may appear in a concrete MAS.

Meta-models have been used to specify AORE notation, like in [Dardenne et al., 1993], where KAOS elements were specified using a meta-model. Another referred work in the MAS domain is the AAlaading framework or AGR (Agent-Group-Role) [Ferber and Gutknecht, 1998]. In that paper, authors show how to model organizations of agents using meta-models. That research has evolved into the MADKIT platform [MADKIT, 1999]. An example of how this meta-model integrates with MADKIT can be found on [Gutknecht, O.,Ferber J., Michel, 2001]. MESSAGE [Caire et al., 2001] proposes a meta-model for MAS that bases on engineering practices. As it is presented in detail in this book, it will be enough to say that its meta-model can be accessed at [EURESCOM P907 consortium, 03], where there are also prototypes and examples of specifications. Following the steps of MESSAGE, readers may also consult INGENIAS [Gomez-Sanz and Pavon, 2003]. INGENIAS official web-site is [GRASIA! research group, 2003]. It also provides examples and Java-based development tools. Finally, meta-modelling is applied too in [Knublauch and Rose, 2002]. [Knublauch and Rose, 2002] presents a visual modelling tool, AGILShell, and a notation to specify MAS. According to authors, this notation is specially suited for the

design of agents that support the information flow between humans in existing work groups.

3. Design

According to conventions in software engineering [Pressman, 1982], analysis refers to what has to be done, whereas design determines how it could be done. Design also supplements analysis with information that deals with implementation choice. So design aims at producing concrete instructions that allow programmers or tools to generate a system that satisfies analysis requirements.

Having selected a concrete analysis method influences the kind of design to be performed. Clearly, some of the works introduced in the previous section already include design concerns. So it would seem natural to continue the design using the same method. However, analysis does not compromise that all. Certainly, an experienced developer can choose any of the previous analysis approaches and still have freedom to selected another different kind of design. The link between design and implementation is harder to broke.

This section introduces two main tendencies in an agent oriented design. Some works suggest that design mainly consists in a refinement of diagrams [DeLoach, 2001] [Collis and Ndumu, 1999]. These works relay on a specific *development environment* that translates diagrams to code. A development environment is a tool or set of tools capable of performing tasks required by a developer in order to build a software system. Usually, a development environment also defines a set of components, or a framework, to be used in the final system. This reduces the set of decisions to be taken during design. At the same time, it also makes the design less flexible, because there are parts that cannot be changed easily. Another trend is not to obviate these decisions and face the whole development from scratch, selecting adequate frameworks and libraries. [Carole Bernon, Marie-Pierre Gleizes, Sylvain Peyruqueou, 2002] [Kinny et al., 1997] belong to this kind of works. Indeed, this is more flexible, but also harder to realize since it requires quite more experience than the other alternative.

3.1 Design with an Development Environment

Development environments for MAS building are oriented towards rapid prototyping. First development environments for MAS generation combined a graphical front-end and a MAS framework. The purpose of this front-end was to facilitate the MAS framework configuration. The resulting prototype was an instantiation of this framework. These en-

vironments for MAS generation were ZEUS [Nwana et al., 1999] and AgentBuilder [IntelliOne Technologies, 1999]. ZEUS includes ontologies definition, communication among agents, and planning capabilities. It is a really good tool if a developer wants to create MAS quickly and experiment with their features. AgentBuilder is supposed to be the evolution of the agent programming language Agent0 [Shoham, 1993]. Semantics of the models bases on the original Agent0 programming language. As ZEUS, it includes a visual editor, simulator, and debugger. Unlike ZEUS, it gives developers more control of the development environment, enabling developers to add new customized modules, called *Project Accesory Classes* (PACs), or connecting agentbuilder agents with legacy applications.

These environments are adequate for rapid prototyping. Nevertheless, there are two risks on using them. First, this kind of tools may accelerate the development, but there is no guarantee that the resulting prototype will satisfy initial requirements. The problem may not appear at the beginning, but later on, when development needs become more strict. Besides, underlying frameworks of these tools cannot be modified easily. Usually, their code is not documented, so changing one component is risky. It may cause a general crash of the whole application. Second, developers depend on the authors of the tools to have updated versions with less bugs or improved functionality. By applying the tool to different domains, bugs are likely to appear. Testing of these tools cannot be exhaustive enough to ensure bug-free applications, specially when they are not commercial. Therefore, having regular updates of these environments is fundamental. The conclusions is that a rapid prototyping tool may not be the solution to all kinds of developments, specially if there is few budget available to buy good commercial tools, such as AgentBuilder.

Then, should rapid prototyping be discarded? The answer is no. There are other tools that propose a different way of generating prototypes. The proposal is to decouple the development environment and the predefined prototype. These tools support analysis and design and can produce code into any language. This is the case of agentTool [Multiagent and Cooperative Robotics Lab, 2000], INGENIAS IDE [GRASIA! research group, 2003], PASSI [Burrafato and Cossentino, 2002], and [Bergenti and Poggi, 2001].

The two first act as translators of the specification language they use to a concrete implementation language. In the design, agentTool uses state machines notation, UML sequence diagrams and some variants of class diagrams to represent internal components of agents and the agents themselves. As a meaningful feature of agentTool, it uses a protocol ver-

ification tool, named SPIN [Holzmann, 1991], that prevents deadlocks. INGENIAS IDE, in the line of MESSAGE, uses a language built of common elements in MAS specifications (agents, tasks, resources, organizations, ...) and a hierarchy of relationships that may appear among them. The result is different from UML, though it is constructed in a similar way. In a INGENIAS design, a developer centers on taking analysis elements and enhances the specification with more details and diagrams that illustrate how it could be done. Like UML, INGENIAS IDE uses incremental development techniques.

The two last, PASSI and [Bergenti and Poggi, 2001], propose reusing existing UML design tools and complementing them with agent code generation facilities. These tools also decouple the graphical front-end from facilities to translate a graphical notation to pieces of low level code. Later, these pieces are put together with other existing pieces of low level code. In the case of [Bergenti and Poggi, 2001] documentation is available. However, PASSI does not provide enough information about its approach. In both cases, developers perform design using UML notation in a UML compliant tool. This notation is marked up with stereotypes, the UML extension mechanism to type classes, so that developers can relate to a class titled *Agent* with a piece of software that will be generated later. The approach seems promising since it is a quick method for integrating agent techniques with UML, apart of AUML.

Though it is not a development environment, DESIRE [Brazier et al., 1997] deserves a few lines. Probably, it is one of the most mature works in this field, due to the number of related papers and training courses [IIDS, 2003]. DESIRE proposes a method to build agents based on the recursive composition of interconnected tasks. DESIRE is supported by a theory on the operation of the framework, a method of development, and tools to facilitate a development with this framework [Brazier et al., 2002]. There are examples of the application of DESIRE to different domains, like a scheduler for appointments in a call center [Brazier et al., 1999] or the diagnosis of failures in a fridge [Brazier et al., 2002]. Both papers show in detail how a design takes place in this approach.

3.2 Design without a Development Environment

Not using a development environment means that developers have to work more choosing proper theories, methods, and software. To help developers in selecting and applying them into their systems, it would be a good idea to review works that have made this effort, already. For instance, [Massonet et al., 2002] considers how to translate MESSAGE/UML diagrams to a concrete target agent platform, JADE, where

control of agents are expert system shells. For more detailed examples, readers may consult [Carole Bernon, Marie-Pierre Gleizes, Sylvain Peyruqueou, 2002], [Caire et al., 2001], and [Kinny et al., 1997] which correspond to different approaches to this problem: object-oriented, agent-oriented, and formal methods based. These are works that provide examples of use and that integrate results in existing research in agents. There are others that deserve being mentioned, but the lack of space prevents a serious review. Readers are invited to review chapters in this book to find additional information.

Reading these works, developers will find out that there are too many concerns to take into account. Among others, authors of this chapter highlight the selection of an agent model, agent architectures for models of agent, code distribution issues, agent features design, agent platforms, and MAS frameworks.

Though relevant, the problem of selecting an agent model is not addressed here. As it was said in the analysis section, readers are invited to consult [Huhns and Singh, 1997] to get a general picture of what features provide different representations. To complement this view, [Nwana, 1996] contains a huge collection of references to developments of agents in many domains. Developers can obtain examples of to know how to develop agents for different purposes.

To introduce other aspects, this paper provides three sections. The first deals with agent architectures. The second with issues related with the distribution of agents, MAS or internal components among different nodes in a network. The third presents research works that addresses theory and implementation of different agent features. The last one collects recommendations of agent platforms and frameworks for MAS design.

Agent Architectures. Why an agent architecture? Because an architecture shows how to put together different pieces of software and make them interact. Here, an agent architecture would provide a framework for realizing different features that researchers require from agents. In this domain, agent architectures have been defined in many ways. Specially, when research on agent started, agent architectures, like IRMA [Bratman et al., 1988], were defined using flows of data among interconnected boxes whose functionality was explained in natural language. There are logical definitions of agents, like those expressed as tuples of functions, as in [Wooldridge, 1992]. Each function defines a particular aspect of the agent, like belief revision functions that determine in each state what beliefs are correct. Layered definitions are also frequent, like [Kendall and Malkoun, 1996]. In a layer definition, sensory

inputs come through lower layers and induce reactions which propagate upwards. When an upper layer wants to perform an action in the environment, the process is the opposite: downwards propagation. Each layer inputs the output of lower layers, and viceversa. Finally, there are also definitions biased by object-oriented approaches that define systems, subsystems, their interfaces and how they are interconnected, like agents described in [Garijo et al., 1998]. This kind of definition is very useful towards implementation, since it already identifies the interfaces of components, the number of existing components, and their purpose. In this direction, software engineers also use Architecture Definition Languages (ADL). Reviews on ADL can be found in [Clements, 1996] and [Medvidovic and Taylor, 1997]. An ADL expresses at a high level what subsystems exist and how they connect. Let us notice that UML v. 2.0 is being considered as a kind of ADL. This would be very interesting, since it would facilitate the integration of an *architectural view* in the agent approaches close to UML.

In an effort of classifying existing agent architectures, [Wooldridge and Jennings, 1995a] suggested three paradigms: deliberative architectures, reactive architectures, and hybrid architectures. *Deliberative* stands for *thinking before doing*. These architectures usually have a symbolic representation of knowledge and provide mechanisms to decide actions upon it. Though flexible, these architectures have an important drawback: reasoning mechanism consume too much time. Perhaps when the agent decides what to do, it may be too late. *Reactive* means that there is no reasoning on deciding what to do next, just associations of inputs and outputs, like *Should happen A, then do B*. These architectures decide what to do next very fast, but chosen action may not be the best one. Finally, *hybrid architectures* are architectures that share deliberative and reactive features. [Wooldridge and Jennings, 1995a] contains representative examples of architectures of each kind that need not to be mentioned again. Just to add a couple of references, this section mentions some recommendable examples of deliberative, SOAR [Laird et al., 1999] and Cougaar [DARPA, 2003], and a hybrid architecture,INTERRAP [Muller, 1996].

SOAR [Laird et al., 1999] derived from the original work of A. Newell [Laird et al., 1987] is an important deliberative architecture. There have been applications of SOAR ranging from modelling human behavior in urban combat till players in first-person-shoot-em-up games. Cougaar [DARPA, 2003], according to their experiments, is may be the most stable agent architecture available nowadays. There are development manuals and examples of developments available at its web site.

INTERRAP [Muller, 1996] shows in detail how this kind of agent is built and used to solve concrete problems. Rather than software, INTERRAP provides the experience of the developer in how a determined organization of components may increase software reuse from one domain problem to another.

The list of agent architectures may continue for pages and pages. Of course, it is recommendable knowing at least representative examples. To save some extra reading, [Muller, 2003] gives rules of thumb to help selecting a suitable agent architectures. These rules are obtained after a study of several application domains and their key features.

Distribution of components. Distribution of agents across a network, parts of a MAS, or just internal components of an agent is a decision that may take place at this stage. As an example of how to deal with distribution of agents, readers can consult [Gervais and Muscutariu, 2001]. As an alternative to distribution, agents can use mobility and forget about where they are. However, an agent can move from one node to a destination node in a network if and only if there is an agent platform, or similar, installed in the destination node. So, who decides which nodes implement what agent platform so that it all works? Deployment is an important part of a specification. The problem is not new at all. In fact, there is notation in UML to deal with this problem and show how the final system should be deployed.

In any case, distribution of agents among different machines, physical or virtual, implies taking into account the issues about how communication may take place, what is being communicated, and how this communication can be used to organize a system behavior.

- **Communication technologies**. Representative technologies that facilitate communication between components are: shared spaces of tuples, emphRemote Procedure Call (RPC), and message passing. The first is a repository of information where several processes are connected and read/write information. An extended, and free, implementation of this technology is Java Network Interfaces (JNI) from Java. The second is based on the existence of a midleware acting as intermediary among two processes. In this variant, a process implements an API, which is offered remotely to other processes in any machine through this middleware. The main reference in this technology is CORBA [OMG, 2000a], a standard ported to most programming languages and Operative Systems. CORBA provides many more things than RPC, but this is not the place for such a discussion. Another middleware offering similar features as CORBA is .NET [Microsoft, 2002] which is recently born,

compared to CORBA, and based on Microsoft platforms. Also, the famous Remote Method Invocation (RMI) available in Java, that supports serialization and transmission of objects through the network. Finally, message passing, though not new at all, perhaps the most frequently used in agent research. It may relay on any of the previous communications technologies since what it defines is asynchronous high level communication among two processes. Messages themselves are described with *Agent Communication Languages* (ACL).

- **Agent Communication Languages**. An ACL describes the format and semantics of messages interchanged among two or more agents. Interpreting properly an ACL requires more than simply parsing the message and extracting data, as remarks [Genesereth and Ketchpel, 1997], since messages have an implicit semantics that detail what kind of reaction is expected in the receiver. There are nowadays two main ACL: KQML [KQML, 1999] [Labrou and Finin, 1997] and FIPA-ACL [FIPA, 2003a]. The first established the bases of current FIPA-ACL, identifying a set of speech acts to be used in agent communication. However, current research focuses in FIPA-ACL, that synthesizes the best of KQML together with other aspects of agent communication, such as standard protocol definition, content languages, and ontologies. Readers interested in the semantics of FIPA-ACL are invited to review its specification, where semantics are expressed using modal logics. Both KQML and FIPA-ACL have several implementations accesible from [KQML, 1999] and [FIPA, 2003b].

- **Ontologies**. An ontology determines allowed terms in the content of a message, as well as concrete semantics and relationships with other elements of the ontology. Specialized languages to define ontologies are Resource Description Framework RDF [W3C, 03], a classic one, and DAML [DARPA, 2001], the unofficial successor of RDF nowadays. To handle ontologies, there are tools such as those available at available at [DARPA, 2001] and Protegè [Stanford Medical Informatics, 2003], one of the most famous. [Stanford Medical Informatics, 2003] contains libraries of ontologes as well as tutorials, papers, and examples. Also, there are extensions to the JADE agent platform to use ontologies created with Protegè.

- **Coordination**. Coordination languages provide description of how interaction should perform over the time. The importance of coordination in MAS does not need justification. According to

[Gelernter and Carriero, 1992], a system can be built out of a computational model and a coordination model. The first deals with sequences of instructions to be executed without interruption. The second attends to how these pieces appear together so that the system satisfies its initial requirements. To achieve such decoupling of aspects, computation vs. coordination, and their later integration, coordination languages propose both a language to express the coordination and frameworks to support these languages. For a general discussion about relationships between computation and coordination, pros and cons of different integration solutions, readers can consult [Gelernter and Carriero, 1992]. In this domain, a main reference is the Linda language [Carriero and Gelernter, 1989]. It defines how coordination can be defined when the communication is performed over a shared tuple space. For a complete review of relevant coordination languages, it is recommended to read [Papadopoulos and Arbab, 1998]. But, how does a developer apply these languages in a MAS? What is different between conventional coordination languages and MAS is the computational model used by agents, which, in general, is more complex. To help in the adaptation of these concepts, readers can consult [Bergenti and Ricci, 2002]. This paper shows how coordination takes place with coordination languages like the previous, with protocols and through the semantics of ACLs. Besides these references, readers are invited to consult the chapter titled *coordination infrastructures* that appears in this book.

For instance, JADE uses, by default, RMI as internal communication facilities, defines wrappers for FIPA-protocols and other facilities to decouple computation from interaction, and includes components to define ontologies. In this sense, JADE is very complete. However, a designer must know that it is possible to generate other kinds of agent-based systems by reusing existing technology.

Agent features. Researchers associate agents with certain features like autonomy, social ability, or intelligence. This section gathers references to research work in designing these features. Why considering these elements in the design section? Because agent abilities are related with agent models and, as it was mentioned before, agent model selection had been postponed to design.

Autonomy, as intelligence, is a term that it is hard to define since it involves philosophical considerations. According to dictionaries, it can be understood as *freedom of will*. However, [H. Hexmoor, C. Castelfranchi, 2003], a kind of survey on autonomy, shows that there are many forms

of autonomy and different ways of understanding it. Among others, it refers to *adjustable autonomy*, the user of an agent decides whether it is autonomous or not, and *behavioral autonomy* as the agent's capacity to be original and not guided by outside source. But, how to achieve or describe autonomy?

- [Hexmoor, 2001] defines a predicate calculus account of autonomy using a *Believes Desires Intention* model. This representation clarifies the notion of autonomy through the concept *situated autonomy*.

  > *Situated autonomy* is an agent's stance, as well as the cognitive function of forming the stance, toward assignment of the performer of a goal at a particular moment when facing a particular situation.

- [Luck and d'Inverno, 1995] describes autonomy using Z [Spivey, 1992a] notation. This description assumes that autonomy arises when an agent has a set of *motivations*. Here a *motivation* is:

  > ... any desire or preference that can lead to the generation and adoption of goals and which affects the outcome of the reasoning or behavioral task intended to satisfy these goals.

- [Castelfranchi and Falcone, 1998] proposes to understand autonomy through a theory of delegation. In this paper, an agent is autonomous with respect a task delegated by other agent. Say an agent A has to execute a task demanded by an agent B. Factors to be considered in order to qualify the degree of autonomy of A with respect B according to this delegated task are: how unspecified the task to execute is, who is responsible of checking the state of the task, and what decisions have been delegated.

- [Weiß et al., 2003] introduces a formalism called *Role Norms Sanctions* (RNS) for explicit specifying the autonomy of computational agents. The idea behind RNS is to support developers of agent-oriented systems in precisely stating what an agent as an autonomous entity is allowed, obliged and forbidden to do. RNS is supported with a tool called XRNS, so developers can generate rather easily their definitions.

Artificial Intelligence and Distributed Artificial Intelligence are the main disciplines in studying computational aspects of intelligence. [Russell and Norvig, 1995a] is a rather complete review of modern AI research in this topic. It focus on AI applied to the construction of intelligent agents. With respect DAI, [Ferber, 1999] makes an account of relevant research of DAI in the context of MAS. Also, [Weiß, 1999] contains a

broad review of research in DAI and MAS considering key topics such as distributed problem solving, reasoning, or MAS learning.

As a guideline for reviewing computational intelligence, a broad field, this section follows list of relevant topics in intelligent agents according to [Russell and Norvig, 1995b]. These are planning, Problem Solving Methods, Learning and Reasoning. For a deeper understanding of the topics reviewed here, [Rich and Knight, 1990], [Russell and Norvig, 1995b], [Weiß, 1999], or [Ferber, 1999] should be consulted. Developers looking for software for any of these topics, can go to [CMU, 2003] and download it. This address also contains links to relevant material in AI.

- **Planning**. Incorporating planning capabilities means that an agent will know, without human intervention, how to combine different tasks in order to get a concrete result. There is a lot of research in planning in AI. STRIPS [Fikes and Nilsson, 1971] is one of the seminal works in the area. It provides a planning algorithm and a language to describe tasks. In the agent domain, TAEMS [Decker, 1996] and Generalized Partial Global Planning (GPGP) algorithm [Decker, 1995] must be mentioned. The first provides concepts and notation for the second, which is a distributed planning solution for tasks that assumes that participants in a plan may have only partial information about it. For both works, there are software and examples available at [Multi-Agent Systems Lab, 03]. A more comprehensive list of planning systems and architectures can be found in [Amant, 03].

- **Problem Solving Methods (PSM)**. A PSM describes the reasoning components of knowledge-based systems as patterns of behavior that can be reused across applications [Fensel and Motta, 2001]. Using PSM can make a program autonomous since it provides reusable behaviors to solve concrete problems. There are libraries of PSM and methodologies that integrate them into a development lifecycle. For a general overview of what a PSM is, from a Knowledge Engineering point of view, and what kinds are available, readers can consult [Fensel and Motta, 2001]. For an adaptation of PSM to the agent domain, readers should consult MAS-CommonKADS [Iglesias et al., 1998b], specially the section dedicated to *task model* and *knowledge modelling* which addresses PSM integration. For an extended version, readers can consult original work in [Iglesias, 1998]. Finally, extended surveys on distributed PSM can be found in [Decker et al., 1989] and [Durfee et al., 1989].

- **Learning**. Learning is a key element if the developer wants the agents to improve over the time. There are many kinds of learning techniques. Apart from AI literature, readers can consult the machine learning review [Dietterich, 1998]. That paper contains pseudo-code examples for different kinds of algorithms and studies of their performance and results. In the agent domain, readers can find the chapter [Sen and Weiß, 1999], that contains learning techniques applied to MAS, and [Stone and Veloso, 2000], that offers a collection of different MAS scenarios and a collection of useful learning techniques to apply in each of them.

- **Reasoning**. In developing reasoning mechanisms, logics have proven to be a valuable tool. Nevertheless, reasoning capabilities depend strongly on the kind of knowledge representation. For instance, with a first-order logic representation, there exists algorithms allow to draw conclusions or courses of action. It is not exactly planning, since it deals more with theorem proving techniques and logic inference (backward or forward chaining). To know more about trends in reasoning using logics, readers can consult [Joseph Y. Halpern, Ronald Fagin, Yoram Moses and Vardi, 1995]. For a comprehensive review of knowledge modelling techniques, readers can consult [Devedzic, 1999]. That paper collects examples for each kind of representation as well as references to tools that help in modelling and reusing knowledge.

An agent being social, according to [Wooldridge and Jennings, 1995b], meant that an agent had to interact with other agents using an *Agent Communication Language* [Genesereth and Ketchpel, 1997]. Results referred here are closer to [Huhns and Stephens, 1999] point of view. In [Huhns and Stephens, 1999], being social implies characterizing agents with abstractions from sociology and organizational theory. In principle, this kind of view would facilitate developing MAS using peer-to-peer interaction instead of a server-client paradigm. This section will comment results in two research lines on social aspects: how to make an agent conscious of its relationships with other agents and how to build a society of agents. The first line refers to representation of social dependencies and reasoning upon this information. The second line distinguishes between agent being organized and agents organizing by themselves, also known as *self-organizing*. For an alternative point of view about these aspects, readers are invited to consult [Boissier, 2003]. [Boissier, 2003] presents organizations in detail from an observer point of view, from the designer point of view, and from the perspective of an agent.

[Sichman et al., 1994] introduces the study of social dependencies. This paper indicates, starting from an external description of an agent, how to derive dependencies upon other agents and establishes the foundations of the so-called *social reasoning*. This kind of reasoning is presented in more detail in [Sichman and Demazeau, 2001] as:

> ... a mechanism that uses information about the others in order to infer some new beliefs from the current ones.

This research links with other aspects already reviewed in this paper, such as autonomy or coordination. According to authors, these results had been implemented in two systems: DEPINT [Sichman, 1998], a simulator of micro-societies, and DEPNET [CONTE and SICHMAN, 1995], an open MAS.

In the *agents being organized* trend, the first recommended paper is [Ferber and Gutknecht, 1998]. In this trend there is an organization that can be distinguished as an first class citizen or just appear in form of roles and social dependencies. [Ferber and Gutknecht, 1998] presented a meta-model for organization description where organization appeared as first class citizen. This work evolved into the MADKIT [MADKIT, 1999] platform, that will be reviewed later. A similar organization description appears in MESSAGE/UML [Caire et al., 2001], where organizational entities relate with interaction related or task related entities. GAIA [Zambonelli et al., 2000] describe its organization with three main organizational abstractions: organizational rules to constraint system behavior, organization structures by defining roles, and organizational relationships that determine agent to agent dependence. This notion of the organization is partially shared by *enterprise modelling* [Fox and Gruninger, 1998]. This discipline studies how to create computational representations of businesses, governments, or any other organizational structure. In this discipline, readers can find ontologies to describe MAS organizations as well as tools to simulate organizations or formats to interchange organizational processes definitions. The application of agents in enterprise modelling to simulate or implement business process workflows have appeared with relative frequency in the agent literature. [Stohr and Zhao, 2001] introduces basic workflow terminology, concepts, and related architectures. It is a good first step towards understanding what is the role of agents in workflows. Examples of application of agents to workflow automation are [D. et al., 2000], that shows how to define a workflow using XML and reactive/cognitive agents, and [Judge et al., 1998], that shows how agents can enhance a workflow basic functionality.

In the *agents organize by themselves* trend, the main concept is *self-organization*. Self-Organization approaches assume that agents get or-

ganized without human intervention. This idea is the opposite of the previous one, where organization exists by itself, whereas here it is just a bottom-up construct. This had been discussed deeply in philosophy in relation with the concept of *autopoiesis* and biological systems, readers can check [Whitaker, 2003] and [SOFAQ, 2003] for an introduction and related software. Here, researchers look for *emergent behaviors*, or how to obtain certain patterns of global behavior in a MAS by changing the way some agents interact. As an example, readers can review [Roli, 2002], a work based in cellular automata. A cellular automata are cells in a grid that can only switch on or switch off autonomously depending on the state of their neighbors. More examples of self-organization and emergent behaviors can be found in literature about *Multi-Agent Based Simulation* (MABS) like [Moss, 2000] and [Conte et al., 1998], both gather research trends in MABS. There are some attempts of structuring these *self-organization results* into bodies of knowledge. One of them is ADELFE [Carole Bernon, Marie-Pierre Gleizes, Sylvain Peyruqueou, 2002], a methodology supported by the ADELFE toolkit. This methodology, with a dedicated chapter in this book, integrates *Adaptative Multi-Agent Systems* considerations in the development process so that developers can have some control over this kind of systems. Also, there is *Engineering Self-Organizing Applications* [ESOA, 2003] working group. This group, as its name remarks, researches self-organization applied to agents. At their web site, researchers will find references to work on this kind of systems and an interesting survey of their current results.

A work in the middle of the previous trends is [Esteva et al., 2002]. This work provides methods to obtain emergent behaviors that satisfy the needs foreseen by developers at design time. Readers interested in this work can review the chapter dedicated to *Social Agents Design Driven by Equations* (SADDE) methodology in this book.

Besides organizational approaches, there are other alternatives to manage the behavior of a MAS, like policies. In networks and telecommunication domain, the concept of *policy* is widespread. A *policy* describes a constraint in the behavior of the system. This concept of *policy* would be similar to the *social laws* [Shoham and M., 1995]. The main difference is that a *policy* can be changed in runtime. The application of *policy* to agents has been studied in [Dulay et al., 2001]. This paper shows a language to define policies and a framework based on agents to support them. The work is interesting because it shows researchers how to build a system that support behavior changes in runtime.

Agent Platforms and frameworks. Citing [Pree, 1995], frameworks represent application skeletons for a particular domain. The util-

ity of a framework, then, it is to save the effort in developing by reusing architectures, components or libraries. An *agent platform* can be understood as a set of services that allow agent management and communication. An agent platform may come with shells of agents that developers can reuse for their systems. There are so many that reviewing all existing MAS frameworks in this chapter is not possible. Thus only some of them will be mentioned here. For a broader view, readers are invited to go through other surveys like the list of agent software from [IntelliOne Technologies, 1999] or *software reports* and *technology roadmaps* available at AgentLink [AgentLink, 03]. As developers, adopting tools that follow standards is a wise option. However, in the agent domain, standards do not provide answers to some problems, like what the internal structure of agents is or its control. That is the reason to consider not only standards, but also other proposals:

- Standard. There are two standards MASIF and FIPA. MASIF stands for Mobile Agent System Interoperability Facility (MASIF [OMG, 1999]) standard. It is based on pure CORBA to implement communication among agents and mobility. The official platform for MASIF is Grasshopper [IKV++ Technologies AG, 1998]. FIPA stands for Foundations for Intelligent Physical Agents. It proposes several services similar to CORBA, but centered on defining interaction protocols and communication languages. Also, FIPA services can be implemented over different communication technologies, like RMI, CORBA, or HTTP based. JADE [Bellifemine et al., 2001] [Telecomm Italia LAB, 1999] and FIPA-OS [Emorphia, 2000] implement FIPA standard. Other implementations of FIPA are available at [FIPA, 2003b].

- Non-standard. A frequently referred work is RETSINA [The Intelligent Software Agents Lab, 2000]. This framework deals, among others, with advanced matchmarking capabilities [Sycara et al., 1999] that facilitate locating agents relevant for a certain task. RETSINA also indicates a kind of MAS infrastructure by identifying general tasks to be accomplished in the system, and roles that tend to appear, like information brokers. Readers will find that RETSINA has been applied in several projects and lots of papers are available about different aspects of RETSINA. Also, RETSINA supplies software to create agents in their website.

  MADKIT [MADKIT, 1999] is an extensible framework for agent design, in which the graphical front-end is just an optional plugin; it is based on the AAladin framework already mentioned in the analysis section. It provides a basic kernel of MAS where concrete

functionality can be plugged in. Also it includes graphical tools to launch, monitor, or kill agents.

In case readers are interested in developing agents using AI approaches, a good choice is ABLE [IBM, 2002]. It can define agents in terms of predefined modules that implement several control mechanisms ranging from neural networks to decision trees. It also implements learning algorithms, like inference learning. These components can be interconnected in a graphical front-end or manually by a couple of lines of code. This framework is a good tool if the developer wants to experiment how to control an agent by combining different techniques.

The number of non-standard platforms is appealing. Previous ones have been selected due to their impact and sucess in dealing with different agent features. In any case, the platforms referred here are only a small percent. Readers are strongly advised to consult reviews of agent platforms such as [Ricordel and Demazeau, 2000], that centers on the suitability of agent development tools from a methodological point of view, or [Mangina, 2002], that is an exhaustive survey of MAS development frameworks.

4. Implementation

Implementation is the translation of design concepts to programs compilable to executable code or interpretable. To implement a MAS, the language may be conventional or agent-oriented. This paper assumes that, despite the high level of abstraction of agent oriented languages, they are still programming languages. Therefore, proposing a development using exclusively an agent oriented language, without specialized analysis notations, or without taking into account design concerns, may be affordable in a small development. But, the same approach in a medium size may not be realistic.

This section does not addresses the problems of reusing software (agent platforms, MAS frameworks, development environments). It is assumed that these elements have already been selected in the design and their influence taken into account. So the problem is to select a language to implement these components. If the developer is using some existing software, perhaps there is no choice.

- Declarative languages (Functional and logical based).
  - April [McCabe and Clark, 1995] is a functional language that incorporates communication facilities. Examples of developments with April can be downloaded from [Labs, 03]. Refer-

ence manuals available from *April* web site contain examples of application.

- Concurrent-METATEM [Fisher, 1995b] allows to express temporal logic programs. A previous paper [Fisher, 1994] shows applications of this language and some small examples.
- CLIPS [NASA, 2003] stands for C Language Integrated Production System. It is an expert system shell that can be used to implement the behavior of an agent. The behavior can be expressed using rules, as it is done in some works referred in this chapter. Support tools and manuals for CLIPS can be downloaded from [NASA, 2003]. There are ports to JAVA (JESS [Friedman-Hill, 2003]). JESS also has been used into JADE agents as an alternative to the default JADE behavior definition.
- Mozart [MOZART, 2003] is a multiparadigm language. Concretely, it is a functional concurrent object-oriented language. There have been some enhancements to support mobility. [Van Roy and Haridi, 1999] is a review of some agent-based projects using Mozart.
- Prolog is a pioneer language in logic programming. Its origin is quite complex and many researchers have worked on it. [Cohen, 1988] contains a review of relevant research work involved in the creation of this language. There are many implementations of this language and suitable extensions. [Sadri and Toni, 1999] contains references to extensions of this language to define agents. Each extension provides its own examples of application.
- Lisp [McCarthy, 1978] is one of the first languages in AI. There is a huge collection of libraries and utilities based on Lisp [ALU, 2003]. [ALU, 2003] also contains references to applications of Lisp for agent programming, like LISA [LISA, 2003], a production-rule system implemented in Lisp, or OSCAR [Pollock, 03], an architecture for anthropomorphic agents.

■ Agent-oriented languages. They incorporate concepts common to agent theories but do not provide primitives dealing with concurrence or temporal logic.

- ConGOLOG [Giacomo et al., 2000] is a language based on situation calculus. It is the concurrent version of GOLOG. It is downloadable from [Cognitive Robotics Group, 03], where

development examples are also available. As an example of a development with ConGOLOG, readers can consult [Yves Lesperance, Todd G. Kelley, John Mylopoulos and Yu, 1999] where authors model a mail-order business.

- Agent0 [Shoham, 1993] is the first agent-oriented language. Unfortunately, there are no interpreters available in the Internet. PLACA [Thomas, 1995] is frequently referred as an extension of Agent0 to MAS. It incorporates ideas of planning among several agents. As Agent0, it cannot be found in the Internet.
- AgentSpeak(L) [Rao, 1996] is designed as a definition language for BDI agents. An implementation of AgentSpeak(L) is SIM SPEAK [Machado, 2003] [Machado and Bordini, 2001]. There is an extension named AgentSpeak(XL)[Bordini et al., 2002] that incorporates elements from TAEMS [Decker, 1996] to the original work.
- MAML [Gulyas and Corliss, 1999] is a language to model social simulations of MAS. It is derived from SWARM [Swarm, 2000], another language for MABS. Tutorials, software, and examples can be accessed at [MAML, 2003].
- 3APL [Hindriks et al., 1999] stands for *Abstract Agent Programming Language*. This language has been defined to express control and deliberation in BDI agents. There is software, tutorials, and examples available at 3APL web site [de Boer, 2003].

Of course, a developer can choose any conventional language: structured or Object-Oriented. In these cases, the developer has to rely on libraries that provide basic functionality, like communication facilities. Most of the agent platforms presented in previous sections are implemented using object-oriented paradigm. However, it may be needed to integrate different programming paradigms into a single architecture. In that case, there are three possibilities: wrapping the foreign language, creating mediators among these structures implemented in different languages, or simply rewriting foreign code. Wrapping is the solution to integrate expert system shells and agent architectures, as the JESS and JADE integration. For mediators, a low cost solution is to use middleware, like CORBA. Mediators of components implemented in different languages offer a remote interface, which is accessible using the same facilities. Rewriting should be the last option, since it requires a lot of work.

In combination with these languages, the developer can use other purpose specific languages. There are languages specially designed to cover issues like coordination, knowledge representation, or ontology representation. References to such languages have been already mentioned in the previous section dealing with architectural issues.

5. Testing

Testing enables to identify the existing failures and to check if the code sticks to the specification of the system or at least, if it satisfies the requirements of customers. In this stage, classic software engineering distinguishes between validation and verification. Citing [Boehm, 1984], in the case of validation, an engineer focuses on the question "Am I building the right software?", whereas in the case of verification, the focus is on the question "Am I building the software right?". In other words, verification is concerned with the (formally) checking the internal consistency of specifications, and validation is concerned with checking the specifications' consistency with the stakeholder's intentions.

Research in testing MAS has mainly addressed verification aspects. Validation of MAS has been studied in works related with agent-oriented requirements engineering, like i* and TROPOS. As it was reviewed in the analysis, software requirements are labelled as *goals*. As they appear as first-class concepts of the specification, it is easy to say if some requirement has been considered or not. It is a matter of checking whether a goal has any associated activity or not. This naive approach does not work in all cases. For an example of more complex validation using a AORE approach, readers can consult [Dubois, 1998]. This paper contains an example of validating a system using the specification language Albert II. The paper also describes a tool able to simulate the specification so that clients can see how it should work and perform the validation themselves.

Since there is more literature dealing with verification than validation in the agent domain, verification will be considered in detail in the next section. The last section will be dedicated to study debugging approaches with agents. Debugging is the complementary task of validation and verification. It seems coherent to reference also research works and tools that help developers to find out exactly why the program has failed.

5.1 Verification of MAS

As an introduction to conventional software engineering testing techniques, readers can consult [Pressman, 1982]. The most relevant ones

are *black box* and *white box* tests. The first considers the system as a black box where only its inputs and outputs are known. The second is quite similar, although it assumes a certain knowledge of internals of the black box. In MAS, white and black box testing is rarely published as original research. Though still infrequent, MAS formal verification is more likely to appear.

Formal verification is based on the existence of an specification expressed with a formal language. In the analysis section, there was a review of different languages that could be used for formal specification. The verification itself can be applied anytime. A verification usually demonstrates the correctness of a program with respect to a specification of what it has to do. For instance, based on Petri Nets formalisms, [Xu et al., 2002] determine if a plan performed with the collaboration of several agents can be performed. For this, they input a Petri Nets specification of the plan. On the other hand, to verify communication from AUML diagrams, [Poutakidis et al., 2002] suggest a mapping method from AUML diagrams to Petri Nets, which well-known algorithms of deadlocks detection are associated with.

According to [Wooldridge and Ciancarini, 2000] there is little work in the verification of MAS. Existing works could be categorized into axiomatic approaches and model checking approaches. The first have several variants and all of them can be considered as theorem proving problems. Research in automated theorem proving field started early in the last century. Unfortunately, it requires high skills in logics for those interested in applying it. For an overview of what kinds of theorem proving techniques exist, readers can consult [Bledsoe, 1985], a survey frequently referred in the area. With respect to application of theorem proving in MAS, [Wooldridge and Ciancarini, 2000] cites works in which axiomatic verification is applied to MAS specified with BDI logics and Concurrent Meta-MEM. However, as [Wooldridge and Ciancarini, 2000] notice, a main problem here is how to apply this kind of verification when the BDI principles are implemented with non-logic based languages, such as C++ or JAVA. This is known as the *computational grounding* problem.

Let us also mention the trend called *design by contract* [Meyer, 1992] that consists in defining pre/postconditions and invariants for the methods or procedures of the code and verifying them in runtime. Violating any of them raises an exception. This technique is built-in in last versions of JAVA and can be simulated in C++ [Plosch, R.; Pichler, 1999]. The problem is this technique does not check program correctness, it just informs that a contract has been violated.

Model checking is less fine grained than axiomatic approaches, but also more tractable. It consists in verifying concrete properties that a system must satisfy. The method inputs a model of the system to check and a property definition. Of course, the language in which model and property are defined is relevant. This approach seems to be well accepted by industry. Its difficulty is the identification of the interesting properties to check. Some of them are listed hereafter, extracted from the literature:

- Liveness. The agent always has something to do. Wooldridge [Wooldridge, 1992] names it, according to his model, the *weakly complete* concept: *there should always be at least one applicable action/message available to every agent, whatever its beliefs.* This property is relevant because a developer surely wants that the agents do not get stuck.

- Deadlock free. A MAS, or some agents belonging to it, may fall in different kinds of deadlocks. A deadlock means, in general, that an agent is blocking others and being blocked at the same time. This may happen, for instance, when there are shared resources among agents and an agent requires a resource that the other holds and viceversa, but none of them wants to release them. There have been thorough deep studies of deadlocks in the concurrent programming, operating systems and telecommunication literature. To have a deeper knowledge on deadlocks, theory and practice, it is very recommended reading the classic [Coffman et al., 1971]. This paper focuses on deadlocks mixing both theory and practice on detection and prevention.

- Another interesting property is that a task must not take the system to an undesirable state. This is a property required in the situation calculus approach of ConGOLOG [Giacomo et al., 2000]. Unlike other properties, ConGOLOG itself takes care of this aspect. For a developer, this property has a high value, since it ensures that state of the world always changes according to the desires of the developer. However, in real systems, this property is very hard to ensure.

Some free software is available to perform model checking and theorem proving. [Bowen, 2003] contains references to this kind of software. As an example of how to apply model checking to MAS, apart of those cited by [Wooldridge and Ciancarini, 2000], readers can consult agenTool [DeLoach, 2001] and MABLE [Wooldridge et al., 2002]. Both use the free *SPIN* model checker [Holzmann, 1991]. AgenTool uses SPIN

allowing to identify deadlocks in agent interaction. MABLE is a programming language enabling to include asserts in the code. These are named *claims*. SPIN is used to verify their truthfulness. It must be said that MABLE is different from the *design by contract* solution, that is verified only in runtime, whereas SPIN verifies during compilation.

Model checking, in principle, requires translating the system into a model specified with a concrete language. SPIN uses as input a language named PROMELA [Holzmann, 1991]. However there are works that suggest that this is not a compelling condition. For instance, [Visser et al., 2000] have studied how, from source code in c++, extract a model of the behavior of the agent, by adding some instructions to the original code, and then perform the model checking.

5.2 Debugging MAS

What if something goes wrong? Researchers in debugging distributed systems give some answers. Debugging MAS is similar to debug open distributed systems or concurrent systems. As developers know, an important problem of distributed systems is that there is too much information to analyze. There is literature that deals with this problem, like [Garcia-Molina, H., Germano, 1984] or [Joyce et al., 1987]. [Joyce et al., 1987] describes the construction of a monitoring system and how collected information can be presented to the user. Benefits of visual representations for debugging is also discussed in [Baecker et al., 1997], where authors propose different kinds of visual and audio presentations of source code and insights of a program. In other trend, [Garcia-Molina, H., Germano, 1984] proposes generation and later analysis of traces of programs.

In the agent domain, there are few tools that can help to test and then debug a system:

- ZEUS incorporates visual debuggers to view internal state of agents. Authors of ZEUS extend their ideas in [Ndumu et al., 1999] proposing internal and external inspectors for a MAS.
- JADE has a *sniffer agent* that shows in a separate GUI what ACL messages are exchanged by the agents.
- MadKit [Gutknecht, O.,Ferber J., Michel, 2001] uses graphical tools and introspection on agent code to discover at runtime groups and roles, references to other agents, and other direct manipulation of these structures.

The drawback of these tools is that they must be used since the beginning of the project. So, what if the framework or the agent platform is

none of these? Then, the developer should consider using the techniques commented at the beginning of the section.

6. More information

To broader the vision of the field, readers are invited to consult other surveys. As a recommedation, it is suggested [Jennings et al., 1998], [Nwana and Ndumu, 1999], and [Weiß, 2003] (the latter served as a main source of inspiration for this chapter).

Current trends in software agents are well reflected in the AgentLink website [AgentLink, 03]. Under AgentLink cover, there are regular publications like the *agentlink roadmaps*, published each year, and special interest research groups on different topics, like the *Methodologies and Software Engineering for Agent Systems* (MSEAS) group, which is specialized in methodologies. AgentLink also sponsors *European Agent Systems Summer School* (EASSS), a school whose proceedings contain tutorials about relevant agent topics.

The Object Management Group, responsible of UML and CORBA standards, has a special interest group on agents [OMG, 2003] with links to projects and documents.

With respect to conferences, let us mention *Autonomous Agents and Multi-Agent Systems* (AAMAS) conferences, which are of very high quality and the *AOSE workshop*, focused on software engineering for agents. There are also chapters dedicated to MAS research in most conferences on Artificial Intelligence, like ICAI, ECAI, or IBERAMIA, whose proceedings appear in catalogues of relevant publishers.

7. Conclusions

This chapter has introduced briefly research results that can help developers and MAS researchers to create MAS. It has surveyed which tools, software libraries, frameworks, theories, and methods are available today for developers. Indeed, having so many results is good news since a developer can produce a MAS with less effort than years ago. However, there are still important gaps and questions, like how to jump from agent theories to MAS implementation, which are the consequences of selecting a concrete agent architectures, how to reuse existing MAS development experience in other developments, or what concepts are needed to tackle with each aspect of a MAS. The agent community is making a huge effort to answer these and others questions, and this is not a trivial task at all. So the best way to finish this chapter is simply congratulating researchers for the work done and encouraging them to keep on contributing to this field.

Acknowledgements

Gerhard Weiß greatly acknowledges support by the German National Science Foundation (DFG) under contract Br609/11-2. Jorge J. Gomez Sanz acknowledges support by Spanish Ministry of Science and Technology under grant TIC2002-04516-C03-03 and Ruben Fuentes and Juan Pavon for reading this chapter and providing useful comments.

References

AgentLink (03). The european network of excellence for agent based computing. http://www.agentlink.org.

ALU (2003). Association of lisp users. http://www.alu.org.

Amant, Rob St. (03). Planning resources at the north carolina state university. http://www.csc.ncsu.edu/faculty/stamant/planning-resources.html.

AUML Team (03). Agent UML web site. http://www.auml.org.

Baecker, Ron, DiGiano, Chris, and Marcus, Aaron (1997). Software visualization for debugging. *Communications of the ACM*, 40(4):44–54.

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

Bellifemine, Fabio, Poggi, Agostino, and Rimassa, Giovanni (2001). Jade: a fipa2000 compliant agent development environment. In *Proceedings of the fifth international conference on Autonomous agents*, pages 216–217. ACM Press.

Bergenti, Federico and Poggi, Agostino (2001). A development toolkit to realize autonomous and interoperable agents. In *Proceedings of the fifth international conference on Autonomous agents*, pages 632–639. ACM Press.

Bergenti, Federico and Ricci, Alessandro (2002). Three approaches to the coordination of multiagent systems. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 367–372. ACM Press.

Bledsoe, Lawrence J. Henschen W. W. (1985). What is automated theorem proving? *Journal of Automated Reasoning*, 1(1):23–28.

Boehm, B. W. (1984). Verifying and validating software requirements and design specifications. *IEEE Software*, 1(1):75–84.

Boissier, Olivier (2003). Master web intelligence: Organizations. http://www.emse.fr/~boissier/enseignement/sma03/pdf/organisation.4pp.pdf.

Bordini, Rafael H., Bazzan, Ana L. C., de O. Jannone, Rafael, Basso, Daniel M., Vicari, Rosa M., and Lesser, Victor R. (2002). Agentspeak(XL): efficient intention selection in BDI agents via decision-theoretic task scheduling. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1294–1302. ACM Press.

Bowen, Jonathan (2003). Formal methods resources. `http://www.afm.sbu.ac.uk`.

Bratman, M. E. (1987). *Intentions, Plans, and Practical Reason*. Harvard University Press.

Bratman, M. E., Israel, D., and Pollack, M. (1988). Plans and resource-bounded practical reasoning. *Journal of Computational Intelligence*, 4(4):349–355.

Brazier, F., van Langen, P., Treur, J., Wijngaards, N., and Willems, M. (1994). Modelling a design task in DESIRE: the VT example. Technical Report IR-377, Universiteit Amsterdam, Department of Mathematics and Computer Science, Vrije, Amsterdam.

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

Brazier, F.M.T., Jonker, C.M., Jungen, F.J., and Treur, J. (1999). Distributed scheduling to support a call centre: a co-operative multi-agent approach. *Applied Artificial Intelligence Journal*, 13. Special Issue on Multi-Agent Systems.

Brazier, F.M.T., Jonker, C.M., and Treur, J. (2002). Principles of component-based design of intelligent agents. *Data and Knowledge Engineering*, 41.

Burrafato, P. and Cossentino, M. (2002). Designing a multi-agent solution for a bookstore with the PASSI methodology. In *Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002)*, Toronto, Ontario, Canada. CEUR-WS.

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

Caire, G., Leal, F., Chainho, P., Evans, R., Garijo, F., Gomez-Sanz, J. J., Pavon, J., Kerney, P., Stark, J., and Massonet, P. (2001). Agent oriented analysis using MESSAGE/UML. In G. Weiß P. Cianciarini, M. Wooldridge, editor, *Agent-Oriented Software Engineering II: Second International Workshop, AOSE 2001, Montreal, Canada, May 29, 2001. Revised Papers and Invited Contribution*, LNCS 2222. Springer Verlag.

Carole Bernon, Marie-Pierre Gleizes, Sylvain Peyruqueou, Gauthier Picard (2002). ADELFE, a methodology for adaptive multi-agent systems engineering. In Tolksdorf, P. Petta R. and Zambonelli, F., editors, *Engineering Societies in the Agents World III: Third International Workshop, ESAW 2002*, number 2577 in LNCS, pages 156 – 169, Madrid, Spain. Springer Verlag.

Carriero, Nicholas and Gelernter, David (1989). Linda in context. *Communications of the ACM*, Volume 32(Issue 4):444–458.

Castelfranchi, C. and Falcone, R. (1998). Towards a theory of delegation for agent-based systems. *Robotics and Autonomous Systems*, 24:141.

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

Clements, Paul C. (1996). A survey of architecture description languages. In *Eighth International Workshop on Software Specification and Design*.

CMU (2003). CMU artificial intelligence repository. `http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/0.html`.

Coffman, E. G., Elphick, M., and Shoshani, A. (1971). System deadlocks. *ACM Computing Surveys (CSUR)*, 3(2):67–78.

Cognitive Robotics Group (03). Concurrent GOLOG. `http://www.cs.toronto.edu/cogrobo/systems.html`.

Cohen, Jacques (1988). A view of the origins and development of prolog. *Communications of the ACM*, 31(1):26–36.

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

Collis, J. C. and Ndumu, D. T. (1999). *The Role Modelling Guide*. Applied Research and Technology, BT Labs.

Conte, R., Gilbert, N., and Simao Sichman, J. (1998). MAS and social simulation: A suitable commitment. In Sichman, Jaime S., Conte, Rosaria, and Gilbert, Nigel, editors, *First International Workshop on Multi Agent Based Simulation '98*, volume 1534 of *Lecture Notes in Computer Science*, pages 1–9. Springer Verlag.

CONTE, Rosaria and SICHMAN, Jaime Simao (1995). Depnet: How to benefit from social dependence. *Journal of Mathematical Sociology*, 20(2-3):161–177.

CSAI LAB (03). a Process for Agents Societies Specification and Implementation (PASSI). `http://www.csai.unipa.it/passi`.

D., D. Walshe, Kennedy, J., Corley, S., Koudouridis, G., Laenen, F.V., Ouzounis, V., Garijo, F., and Gomez-Sanz, J. (2000). Eurescom p815: An interoperable architecture for agent-oriented management. In Arab-

nia, Hamid R., editor, *IC-AI'2000 Proceedings*, volume I. International Conference on Artificial Intelligence 2000, CSREA Press.

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

Darimont, R., Delor, E., Massonet, P., and van Lamsweerde, A. (1997). GRAIL/KAOS: an environment for goal-driven requirements engineering. In *Proceedings of the 19th international conference on Software engineering*, pages 612–613. ACM Press.

DARPA (2001). Darpa agent markup language. `http://www.daml.org/language`.

DARPA (2003). Cognitive agent architecture. `http://www.cougaar.org`.

de Boer, Frank S. (2003). An abstract agent programming language (3apl). `http://www.cs.uu.nl/3apl`.

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

Decker, Keith S. (1996). Task environment centered simulation. In *Simulating Organizations: Computational Models of Institutions and Groups*. AAAI Press/MIT Press.

Decker, S. K. (1995). *Environment Centered Analysis and Design of Coordination Mechanisms*. Ph.d. thesis, Department of Computer Science, University of Massachusetts.

DeLoach, S. (2001). Analysis and Design using MaSE and agenTool. In *Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conferece (MAICS)*, Miami University. Miami University Press.

Demazeau, Y. (1995). From cognitive interactions to collective behaviour in agent-based systems. In *Proc. European Conference on Cognitive Science*, pages 117–132, Saint-Malo, France.

Depke, Ralph, Heckel, Reiko, and Kuster, Jochen M. (2001). Improving the agent-oriented modeling process by roles. In *Proceedings of the fifth international conference on Autonomous agents*, pages 640–647. ACM Press.

Devedzic, V. (1999). A survey of modern knowledge modeling techniques. *Expert Systems with Applications*, 17:275.

Dietterich, Thomas G. (1998). Machine-learning research: Four current directions. *The AI Magazine*, 18(4):97–136.

d'Inverno, M. and Luck, M. (1996). A formal view of social dependence networks. In Zhang and Lukose, editors, *Distributed Artificial Intelligence Architecture and Modelling: Proceedings of the First Australian*

Workshop on Distributed Artificial Intelligence, volume 1087 of *LNAI*, pages 115–129. Springer-Verlag.

d'Inverno, M., Hindriks, K. and Luck, M. (2000). A formal architecture for the 3APL agent programming language. In *ZB2000: Formal Specification and Development in Z and B - 1st International Conference of B and Z Users*, volume 1878 of *LNCS*, pages 168–187. Springer Verlag.

Dubois, E. Du Bois, P. Dubru, F. and Petit, M. (1994). 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.

Dubois, P. Heymans E. (1998). Scenario-based techniques for supporting the elaboration and the validation of formal requirements. Technical Report CREWS Report 98-15, Universite de Namur, Belgium.

Dulay, Naranker, Damianou, Nicodemos, Lupu, Emil, and Sloman, Morris (2001). A Policy Language for the Management of Distributed Agents. In M. J. Wooldridge, G. Weiß, P. Ciancarini, editor, *Agent-Oriented Software Engineering II*, volume 2222 of *LNCS*, pages 84–100, Second International Workshop, AOSE 2001, Montreal, Canada. Springer Verlag.

Durfee, Edmund H., Lesser, Victor R., and Corkill, Daniel D. (1989). Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, 1(1):63–83.

Emorphia (2000). FIPA-OS. `http://fipa-os.sourceforge.net`.

ESOA (2003). Working group engineering self-organising applications of the agentcities project. `http://gaper.swi.psy.uva.nl/esoa/content/main.php`.

Esteva, Marc, de la Cruz, David, and Sierra, Carles (2002). Islander: an electronic institutions editor. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1045–1052. ACM Press.

EURESCOM P907 consortium (03). MESSAGE/UML website. `http://www.eurescom.de/~public-webspace/P900-series/P907/index.htm`.

Fensel, Dieter and Motta, Enrico (2001). Structured development of problem solving methods. *Knowledge and Data Engineering*, 13(6):913–932.

Ferber, J. and Gutknecht, O. (1998). 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.

Ferber, Jacques (1999). *Multi-Agent Systems*. Addison-Wesley.

Fikes, R. and Nilsson, J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4).

FIPA (2003a). FIPA-ACL Specifications. `http://www.fipa.org/repository/aclspecs.html`.

FIPA (2003b). FIPA compliant software. `http://www.fipa.org/resources/livesystems.html`.

Fisher, M. (1994). A survey of concurrent METATEM – the language and its applications. In Gabbay, D. M. and Ohlbach, H. J., editors, *Temporal Logic - Proceedings of the First Intemational Conference*, volume 827 of *LNAI*, pages 480–505. Springer-Verlag.

Fisher, M. (1995a). Representing and executing agent-based systems. In Wooldridge, M. and Jennings, N. R., editors, *Intelligent Agents: Theories, Architectures, and Languages*, volume 890 of *LNAI*, pages 307–323. Springer-Verlag.

Fisher, M. (1995b). Representing and executing agent-based systems. In Wooldridge, M. J. and Jennings, N. R. editors, *Intelligent Agents*, volume 890 of *LNAI*, pages 307–323. Springer-Verlag.

Fox, M.S. and Gruninger, M. (Fall 1998). Enterprise modelling. *AI Magazine*, 19(3):109–121.

Friedman-Hill, E. (2003). Java expert system shell (JESS). `http://herzberg.ca.sandia.gov/jess`.

Garcia-Molina, H., Germano, F. (1984). Debugging a distributed computer system. *IEEE Transactions on Software Engineering*, SE-10(2):210–219.

Garijo, Francisco, Tous, Juan, Matias, Jose M., Corley, Stephen, and Tesselaar, Marius (1998). Development of a multi-agent system for cooperative work with network negotiation capabilities. In Albayrak, Sahin, editor, *Intelligent Agents for Telecommunication Applications*, volume 1437 of *LNCS*, pages 204–219. Springer Verlag.

Gelernter, David and Carriero, Nicholas (1992). Coordination languages and their significance. *Communications of the ACM*, 25(2):97–107.

Genesereth, M. R. and Nilsson, N. J. (1987). *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publisher, Los Altos, CA.

Genesereth, Michael R. and Ketchpel, Steven P. (1997). Software agents. *Communications of the ACM*, 37(7).

Gervais, Marie-Pierre (2003). ODAC : an agent-oriented methodology based on ODP. *Journal of Autonomous Agents and Multi-Agent Systems*, 7(3):199–228.

Gervais, Marie-Pierre and Muscutariu, Florin (2001). Towards an ADL for designing agent-based systems. In Wooldridge, M.J., Weiß, G., and Cianciarini, P., editors, *Agent-Oriented Software Engineering II.*

Second International Workshop, AOSE 2001, Montreal, Canada, May 29, 2001. Revised Papers and Invited Contributions, volume 2222 of *LNCS*, pages 263–277. Springer Verlag.

Giacomo, G. De, Lesperance, Y., and Levesque, Hector J. (2000). Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121:109–169.

Gomez-Sanz, J. and Pavon, J. (2003). Agent oriented software engineering with INGENIAS. In Vladimír Marík and Jörg Müller and Michal Pechoucek, editor, *Multi-Agent Systems and Applications III*, volume 2691 of *LNCS*, pages 394–403, Prague, Czech Republic. 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003, Springer Verlag.

Gomez-Sanz, Jorge J., Pavon, Juan, and Garijo, Francisco (2002). Metamodels for building multi-agent systems. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 37–41. ACM Press.

GRASIA! research group (2003). Ingenias ide. `http://ingenias.sourceforge.net`.

Gulyas, Tamas Kozsik Laszlo and Corliss, John. B. (1999). The multiagent modelling language and the model design interface. *The Journal of Artificial Societies and Social Simulation*, 2(4).

Gurevich, Yuri (1984). Toward logic tailored for computational complexity. *Computation and Proof Theory*, 1104:175–216.

Gutknecht, O.,Ferber J., Michel, F. (2001). Integrating tools and infrastructures for generic multi-agent systems. In *AGENTS01*, pages 441–448.

H. Hexmoor, C. Castelfranchi, R. Falcone (2003). A prospectus on agent autonomy. *Agent Autonomy*, 1(1):1–8.

Hexmoor, H. (2001). A cognitive model of situated autonomy. In *Advances in Artificial Intelligence*, volume 2112 of *LNAI*, pages 325–334. Springer Verlag.

Hindriks, Koen V., Boer, Frank S. De, der Hoek, Wiebe Van, and Meyer, John-Jules Ch. (1999). Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401.

Holzmann, G. J. (1991). *Design and Validation of Computer Protocols*. Prentice-Hall, Englewood Cliffs, New Jersey.

Huhns, M. and Singh, M., editors (1997). *Reading in Agents*, chapter 1, pages 1–23. Morgan Kaufmann Publishers.

Huhns, Michael N. and Stephens, Larry M. (1999). Multiagent systems and societies of agents. In Weiß, G. editor, *Multiagent Systems*, pages 79–120. The MIT Press, Cambridge et al.

IBM (2002). Agent building and learning environment (ABLE). Electronic Citation.

Iglesias, C. (1998). *Definicion de una Metodologia para el Desarrollo de Sistemas Multi-Agente*. Thesis/dissertation, Departamento de ingeniera de Sistemas Telemticos, Universidad Politecnica de Madrid.

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

Iglesias, C., Garijo, M. Mercedes, Gonzalez, J. C., and Velasco, J. R. (1998b). Analysis and design of multiagent systems using MAS-commonKADS. In Singh, M. P., Rao, A., and Wooldridge, M. J., editors, *Intelligent Agents IV*, volume 1365 of *LNAI*. SpringerVerlag.

IIDS (2003). Intelligent Interactive Distributed Systems group. `http://www.iids.org`.

IKV++ Technologies AG (1998). Grasshopper. `http://www.grasshopper.de/index.html`.

IntelliOne Technologies (1999). AgentBuilder. `http://www.agentbuilder.com`.

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

Jennings, N. R., Sycara, K., and Wooldridge, M. J. (1998). A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1:7–38.

Joseph Y. Halpern, Ronald Fagin, Yoram Moses and Vardi, Moshe Y. (1995). *Reasoning About Knowledge*. MIT Press.

Joyce, Jeffrey, Lomow, Greg, Slind, Konrad, and Unger, Brian (1987). Monitoring distributed systems. *ACM Transactions on Computer Systems (TOCS)*, 5(2):121–150.

Judge, D. W., Odgers, B. R., Shepherdson, J. W., and Cui, Z. (1998). Agent-enhanced workflow. *BT Technology Journal*, 16(3):79–85.

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

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

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

Kinny, D., Georgeff, M., and Rao, A. (1997). A methodology and modelling technique for systems of BDI agents. Tech Report 55, Australian Artificial Intelligence Institute, Melbourne, Australia.

Knublauch, H. Holger and Rose, T. (2002). Tool-supported process analysis and design for the development of multi-agent systems. In *AOSE 2002*, LNAI. Springer Verlag.

KQML (1999). *The UMBC KQML Web*. `http://www.cs.umbc.edu/kqml`.

Labrou, Yannis and Finin, Tim (1997). A proposal for a new kqml specification. Technical Report TR CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County.

Labs, Fujitsu (03). April programming language. `http://www.nar.fujitsulabs.com`.

Laird, J. E., Congdom, C. Bates, and Coulter, K. J. (1999). *The SOAR's users manual v.8.2*. The Soar Group, Artificial Intelligence Laboratory, University of Michigan.

Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). SOAR: an architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64.

LISA (2003). Lisp-based intelligent software agents. `http://lisa.sourceforge.net`.

Luck, Michael and d'Inverno, Mark (1995). A formal framework for agency and autonomy. In Lesser, Victor and Gasser, Les, editors, *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 254–260, San Francisco, CA, USA. AAAI Press.

Machado, R. and Bordini, R. H. (2001). Running agentspeak(l) agents on SIM_AGENT. In Meyer, J.J. and Tambe, M., editors, *Intelligent Agents VIII*, volume 2333 of *LNAI*, pages 158–174. Proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001), Springer Verlag.

Machado, Rodrigo (2003). SIM Speak. `http://www.inf.ufrgs.br/~bordini/SIM_Speak`.

MADKIT (1999). *Multi-Agent Development KIT*.

MAML (2003). Multi-agent systems modeling language. `http://www.maml.hu`.

Mangina, Eleni (2002). Review of software products for multi-agent systems. survey, AgentLink.

Massonet, Philippe, Deville, Yves, and Neve, Cedric (2002). From AOSE methodology to agent implementation. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 27–34. ACM Press.

McCabe, F. G. and Clark, K. L. (1995). April - agent PRocess interaction language. In M., Wooldridge and N., Jennings, editors, *Intelligent Agents*, volume 890 of *LNCS*. Springer Verlag.

McCarthy, J. and Hayes, P. J. (1981). Some philosophical problems from the standpoint of artificial intelligence. In Webber, B. L. and Nilsson, N. J., editors, *Readings in Artificial Intelligence*, pages 431–450. Kaufmann, Los Altos, CA.

McCarthy, John (1978). History of LISP. In *The first ACM SIGPLAN conference on History of programming languages*, pages 217–223.

Medvidovic, Nenad and Taylor, Richard N. (1997). A framework for classifying and comparing architecture description languages. In *Proceedings of the 6th European conference held jointly with the 5th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 60–76. Springer-Verlag New York, Inc.

Meyer, B. (1992). Applying design by contract. *IEEE Computer*, 25(10):40–51.

Microsoft (2002). Distributed component object model. `http://www.microsoft.com`.

Moss, Scott (2000). *Editorial Introduction: Messy Systems The Target for Multi Agent Based Simulation*, volume 1979 of *Lecture Notes in Artificial Intelligence*. Springer Verlag.

MOZART (2003). The mozart programming system. `http://www.mozart-oz.org`.

Muller, J. P. (2003). The right agent (architecture) to do the right thing. In *Intelligent Agents V*, volume LNCS 1555, pages 105–112. Springer-Verlag.

Muller, Jorg P. (1996). *The Design of Intelligent Agents, a layered approach*, volume 1177 of *LNCS*. Springer Verlag.

Multi-Agent Systems Lab (03). Multi-agent systems lab. `http://dis.cs.umass.edu`.

Multiagent and Cooperative Robotics Lab (2000). The agentool project. `http://www.cis.ksu.edu/\~{}sdeloach/ai/agentool.htm`.

Mylopoulos, J. and Castro, J. (2000). Tropos: A framework for requirements-driven software development. In *Proceedings of 12th Conference on Advanced Information Systems Engineering (CAISE)*.

NASA (2003). C language integrated production system (CLIPS). `http://www.ghgcorp.com/clips/CLIPS.html`.

Ndumu, Divine T., Nwana, Hyacinth S., Lee, Lyndon C., and Collis, Jaron C. (1999). Visualising and debugging distributed multi-agent systems. In *Proceedings of the third annual conference on Autonomous Agents*, pages 326–333. ACM Press.

Nuseibeh, B. A. and Easterbrook, S. M. (2000). Requirements engineering: A roadmap. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE'00)*, pages 35–46.

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

Nwana, H. S., Ndumu, D. T., Lee, L. C., and Collis, J. C. (1999). ZEUS: A toolkit for building distributed multi-agent systems. *Applied Artificial Intelligence Journal*, 1(13):129–185.

Nwana, Hyacinth S. and Ndumu, Divine T. (1999). A perspective on software agents research. *The Knowledge Engineering Review*, 14(2):1–18.

Odell, J., Parunak, H., and Bauer, B. (2000). Extending UML for agents.

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

OMG (03). XML metadata interchange version 1.1. `http://www.omg.org`.

OMG (1999). Mobile Agent System Interoperability Facility (MASIF). `http://www.fokus.gmd.de/research/cc/ecco/masif`.

OMG (2000a). CORBA 2.4.2 specification. `http://www.omg.org`.

OMG (2000b). Meta Object Facility (MOF). `http://www.omg.org`.

OMG (2000c). Unified Modeling Language Specification. Version 1.3. `http://www.omg.org`.

OMG (2003). OMG Agent Platform Special Interest Group. `http://www.objs.com/agent/index.html`.

Papadopoulos, George A. and Arbab, Farhad (1998). Coordination models and languages. In *The Engineering of Large Systems*, volume 46 of *Advances In Computers*. Elsevier.

Plosch, R.; Pichler, J.; (1999). Contracts: from analysis to C++ implementation. In *Technology of Object-Oriented Languages and Systems*, pages 248–257. IEEE Computer.

Pollock, John L. (03). OSCAR. `http://oscarhome.soc-sci.arizona.edu/ftp/OSCAR-web-page/oscar.html`.

Poutakidis, David, Padgham, Lin, and Winikoff, Michael (2002). Debugging multi-agent systems using design artifacts: the case of interaction protocols. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 960–967. ACM Press.

Pree, W. (1995). State-of-the-art design pattern approaches: An overview. In *Technology of Object-Oriented Languages and Systems (TOOLS 95)*.

Pressman, Roger S. (1982). *Software Engineering: A Practitioner's Approach*. McGraw-Hill Series in Software Engineering and Technology. McGraw-Hill, New York, 6th edition.

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

Rao, Anand S. and Georgeff, Michael P. (1991). Modeling rational agents within a BDI-architecture. In Allen, James, Fikes, Richard, and Sandewall, Erik, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

Rich, E. and Knight, K. (1990). *Artificial Intelligence*. McGraw-Hill.

Ricordel, P.-M. and Demazeau, Y. (2000). 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)*, pages 93–105.

Roli, F. Zambonelli A. (2002). Emergent behaviors in dissipative cellular automata. In *5th International Conference on Cellular Automata for Research and Industry (ACRI 2002)*, Geneva.

Rosenschein, S. J. and Kaelbling, L. P. (1995). A situated view of representation and control. *Artificial Intelligence*, 73(1/2):149–173.

Russell, S. J. and Norvig, P. (1995a). *Artificial Intelligence. A Modern Approach*, chapter AI: Present and Future, pages 842–849. Prentice Hall, Englewood Cliffs, New Jersey.

Russell, S. J. and Norvig, P. (1995b). *Artificial Intelligence. A Modern Approach*. Prentice Hall, Englewood Cliffs, New Jersey.

Sadri, F. and Toni, F. (1999). Computational logic and multiagent systems: a roadmap. Tech report, Compulog Net at DFKI SB.

Schobbens, Pierre-Yves (2003). The ALBERT requirements engineering research group homepage. http://www.info.fundp.ac.be/albert/.

Sen, Sandip and Weiß, Gerhard (1999). Learning in multiagent systems. In Weiß, G. editor, *Multiagent Systems*, pages 259–299. The MIT Press, Cambridge et al.

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

Shoham, Y. and M., Tennenholtz (1995). On social laws for artificial agent societies: off-line design. *Artificial Intelligence*, 73(1-2):231–252.

Sichman, J. S., Conte, R., Demazeau, Y., and Castelfranchi, C. (1994). A social reasoning mechanism based on dependence networks. In *Proc. European Conference on Cognitive Science*.

Sichman, Jaime Simao (1998). Depint: Dependence-based coalition formation in an open multi-agent scenario. *Journal of Artificial Societies and Social Simulation*, 1(2).

Sichman, Jaime Simao and Demazeau, Yves (2001). On social reasoning in multi-agent systems. *Revista Iberoamericana de Inteligencia Artificial*, 3(13):68–84.

Singh, M. (1997). Formal methods in DAI: logic based representation and reasoning. *Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence*, pages 331–376.

SOFAQ (2003). Self-organisation FAQ. `http://www.calresco.org/sos`.

Sommerville, Ian (2001). *Software Engineering*. International Computer Sciences Series. Addison-Wesley, Harlow, UK, 6th edition.

Spivey, J. M. (1992a). *The Z Notation*. Prentice Hall, Hempstead, 2nd edition.

Spivey, J. M. (1992b). *The Z Notation: a reference manual*. Prentice Hall.

Stanford Medical Informatics (2003). Protegè. `http://protege.stanford.edu`.

Stohr, Edward A. and Zhao, J. Leon (2001). Workflow automation: Overview and research issues. *Information Systems Frontiers*, 3(3):281–296.

Stone, Peter and Veloso, Manuela (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383.

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

Sycara, K, Klusch, M., idof, S., and Lu, J (1999). Dynamic service matchmaking among agents in open information environments. *Journal ACM SIGMOD Record , Special Issue on Semantic Interoperability in Global Information Systems*.

Telecomm Italia LAB (1999). Java Agent DEvelopment framework (JADE). `http://sharon.cselt.it/projects/jade/`.

The Intelligent Software Agents Lab (2000). Reusable environment for task-structured intelligent networked agents (RETSINA). `http://www-2.cs.cmu.edu/\~softagents/`.

Thomas, S. R. (1995). The PLACA agent programming language. In Wooldridge, M. J. and Jennings, N. R. editors, *Intelligent Agents*, volume 890 of *LNAI*, pages 355–370. Springer-Verlag, Berlin et al.

Union, International Telecommunication (99 A.D.). ITU 100:formal description techniques (FDT)- specification and description language (SDL). Report.

van der Hoek, Wiebe (2001). Logical foundations of agent-based computing. In Luck, Michael, Marík, Vladimír, Stepánková, Olga, and Trappl, Robert, editors, *Multi-Agent Systems and Applications, 9th ECCAI Advanced Course ACAI 2001 and Agent Link's 3rd European Agent Systems Summer School*, volume 2086 of *LNCS*. Springer.

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

van Lamsweerde, R. Darimont A. and Massonet, Ph. (1995). Goal-directed elaboration of requirements for a meting scheduler: Problems and lessons learnt. In *Proceedings RE'95 - Second International Conference on Requirements Engineering*, pages 194–203. IEEE Computer Society Press.

Van Roy, Peter and Haridi, Seif (1999). Mozart: A programming system for agent applications. In *International Workshop on Distributed and Internet Programming with Logic and Constraint Languages*.

Visser, William, Park, SeungJoon, and Penix, John (2000). Using predicate abstraction to reduce object-oriented programs for model checking. In *Proceedings of the third workshop on Formal methods in software practice*, pages 3–182. ACM Press.

W3C (03). W3C resource description framework (RDF) activity page. http://www.w3.org/RDF/.

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

Weiß, Gerhard (2003). Agent orientation in software engineering. *Knowledge Engineering Review*, 16(4):349–373.

Weiß, Gerhard, Rovatsos, Michael, and Nickles, Matthias (2003). Capturing agent autonomy in roles and XML. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 105–112. ACM Press.

Whitaker, Randall (2003). ACM SIGGROUP: Self-organization, autopoiesis, and enterprises. `http://www.acm.org/sigois/auto/Main.html`.

Wood, M. and DeLoach, S. A. (2001). An overview of the multiagent systems engineering methodology. In Ciancarini, P. and Wooldridge, M. J. editors, *Agent-oriented software engineering. Proceedings of the First International Workshop (AOSE-2000)*, volume 1957 of *LNAI*, pages 207–222. Springer-Verlag.

Wooldridge, M. (1997). Agent-based software engineering. *IEE Proceedings Software Engineering*, 144(1):26–37.

Wooldridge, M., Jennings, Nicholas R., and Kinny, D. (2000). The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 15.

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

Wooldridge, M. J. and Jennings, N. R. (1995a). Agent theories, architectures, and languages: A survey. In Wooldridge, M. J. and Jennings, N. R. editors, *Intelligent Agents*, volume 890 of *LNAI*, pages 1–39. Springer-Verlag, Berlin et al.

Wooldridge, Michael, Fisher, Michael, Huget, Marc-Philippe, and Parsons, Simon (2002). Model checking multi-agent systems with MABLE. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 952–959. ACM Press.

Wooldridge, Michael J. and Jennings, Nicholas R. (1995b). Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10(2):115–152.

Wooldridge, Mike and Ciancarini, P. (2000). Agent-Oriented Software Engineering: The State of the Art. In Ciancarini, P. and Wooldridge, M., editors, *First Int. Workshop on Agent-Oriented Software Engineering*, volume 1957, pages 1–28. Springer-Verlag, Berlin.

Xu, Dianxiang, Volz, Richard, Ioerger, Thomas, and Yen, John (2002). Modeling and verifying multi-agent behaviors using predicate/transition nets. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 193–200. ACM Press.

Yu, E. (1999). Strategic modelling for enterprise integration. In Chen, Han-Fu, Cheng, Dai-Zhan, and Zhang, Ji-Feng, editors, *Proceedings 14 th World Congress of the International Federation of Automatic Control*, IFAC Proceedings Volumes. Elsevier Science Ltd.

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

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

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

Yu, Eric (03a). Eric yu home page. `http://www.cs.toronto.edu/~eric`.

Yu, Eric (03b). GRL web site. `http://www.cs.toronto.edu/km/GRL`.

Yu, Eric and Liu, Lin (03). Organization modelling environment. `http://www.cs.toronto.edu/km/ome`.

Yu, Eric and Liu, Lin (2002). Designing web-based systems in social context: A goal and scenario based approach. In *Advanced Information Systems Engineering*, volume 2348 of *LNCS*, pages 37–51. International Conference, CAiSE 2002 Toronto, Canada, Springer Verlag.

Yu, Eric, Liu, Lin, and Li, Ying (2001). Modelling strategic actor relationships to support intellectual property management. In *Conceptual Modeling - ER 2001*, volume 2224 of *LNCS*, pages 164–178. Springer Verlag.

Yves Lesperance, Todd G. Kelley, John Mylopoulos and Yu, Eric S. K. (1999). Modeling dynamic domains with congolog. In Jarke, A. Oberweis M., editor, *CAiSE'99*, volume 1626 of *LNCS*, pages 365–380. Springer Verlag.

Zambonelli, F., Jennings, N. R., and Wooldridge, M. (2000). Organisational abstractions for the analysis and design of multi-agent systems. In Ciancarini, P. and Wooldridge, M. J. editors, *Agent-oriented software engineering. Proceedings of the First International Workshop (AOSE-2000)*, volume 1957 of *LNCS*, pages 127–141. Springer-Verlag.

Zave, Pamela (1997). Classification of research efforts in requirements engineering. *ACM Computing Surveys (CSUR)*, 29(4):315–321.