An Agent Programming Framework Based on the C# Language and the CLI

A. Grosso, A. Gozzi, M. Coccoli, A. Boccalatte
University of Genova, DIST
Via Opera Pia, 13 – 16145 Genova, Italy
grosso@email.it {gozzi, coccoli, nino}@dist.unige.it

ABSTRACT

The aim of this paper is that of describing the software architecture designed for the implementation of an agent programming framework. Contrary to the widely adopted use of the Java platform in most agent-based solutions and related research activity, the present work has been entirely carried on in the novel .NET framework (Microsoft Corp.) which offers technological solutions able to cope with common problems which often arise when working in other enterprise platforms (both J2EE and Microsoft DNA). In particular this work is based on the C# language and the Common Language Infrastructure. One solution is described to make management of agents easier from a programmer's point of view. As result an agent programming framework is presented, fully exploiting the above concepts.

Keywords
"Agent", "Multi-Agents", "Agent Programming Framework", "Microsoft.NET platform".

1. INTRODUCTION

Agent-based applications and agent systems represent a very robust and theoretically well funded technological paradigm. Despite this, they are not yet widespread at all due to many reasons, one of which is the heavy programming work that still has to be done in order to get efficient and effective agent systems in particular from the point of view of the performance and integration with other applications. In this paper, the authors assume that the reader is familiar with the basic concepts of software agents [Nwa96a][Woo95a] and multi agent systems [Woo99a] for the present work is mainly focused on topics related to internals of the programming tools for such systems. Much attention is dedicated to the software architecture of agent themselves rather then to the architecture of agent systems to be developed.

1st Int.Workshop on C# and .NET Technologies on Algorithms, Computer Graphics, Visualization, Computer Vision and Distributed Computing

February 6-8, 2003, Plzen, Czech Republic.
Copyright UNION Agency – Science Press
ISBN 80-903100-3-6

Distributed systems and distributed programming appear to be the natural field of application for agent technology, due to intrinsic characteristics of agents and multi-agent systems but, on the other hand, modern distributed applications require a very high degree of availability, reliability, and security thus, the development of enterprise solutions will be based on an agent system only in the case it meets all of these basic requirements. After this short introduction, the paper is organized as follows. In the forthcoming Section 2 work already done on agent construction tools will be presented together with problems related to programming agent systems. Basics on the .NET framework will be introduced and discussed in Section 3; relevance will be given to the features exploited in developing this work. Based on the contents of both Section 2 and Section 3, an analysis of the software architecture which implements agents in the proposed environment will follow in Section 4 where the model proposed by the authors will be described; it will also be shown how it can override most common problems which arise with different solutions.

2. AGENT DEVELOPMENT TOOLS

In the recently past years a lot of work has been done related to agent applications both by research institutes and, more and more, by commercial organizations [App02a]. The activity of analysis and evaluation of such software tools is out of the scope of this paper; moreover compiling an up to date list of existing agent development environments can be a very hard work, continuously subject to change. Up to date surveys of available software products for agent systems can be found on many web sites. For example you can refer to the “European Network of Excellence for Agent-Based Computing” [The European Network of Excellence for Agent-Based Computing http://www.agentlink.org] where much information is available. Different tools offer different ability, also depending on the definition of an agent which has been considered while developing the tool. In fact, the definition of an agent is still very discussed in the scientific community and many different, though almost equivalent, definitions can be found (see Section 2.1). The main differentiation can be done based on the observation of different choices, in particular relevant to the specification of the mechanism of communication and the conversation rules as well as the adoption of a standard communication language. Following the adoption of a particular definition, then different models arise.

Agent Definitions

In the agent community, it cannot be found a unique definition of what an agent is or it should be. Different research groups have given different interpretations depending on their research interest and their past experience with software engineering and cultural/technical background. As an example, some widely accepted definitions follow. Pattie Maes (software agents research group at MIT) gives her own definition of the term. She says, “*an agent is a computational system that inhabits a complex, dynamic environment. The agent can sense, and act on, its environment, and has a set of goals or motivations that it tries to achieve through these actions*”. A slightly different point of view comes from Hyacinth S. Nwana [Nwa96b] which, in defining the term agent, says, “*when we really have to, we define an agent as referring to a component of software and/or hardware which is capable of acting exactingly in order to accomplish tasks on behalf of its user. Given a choice, we would rather say it is an umbrella term, meta-term or class, which covers a range of other more specific agent types, and then go on to list and define what these other agent types are. This way, we reduce the chances of getting into the usual prolonged philosophical and sterile arguments which usually proceed the former definition, when any old software is conceivably recastable as agent based software*”. FIPA (Foundation for Intelligent Physical Agents) uses a different definition of agent for all their specifications [FIPA (Foundation for Intelligent Physical Agents), http://www.fipa.org] that is, “*an entity that resides in environments where it interprets "sensor" data that reflect events in the environment and executes "motor" commands that produce effects in the environment. An agent can be purely software or hardware. In the latter case a considerable amount of software is needed to make the hardware an agent*”. On the other hand H. Van Dyke Parunak defines an agent as an “*active object with initiative*”. According to this definition [Par97a], no agent can be independent of its environment, from which it receives inputs to be processed, and it produces output feeding the environment itself. A further point of view is that of researcher groups in the Artificial intelligence community for whom the lemma agent assumes a more peculiar meaning. An agent is meant to be a computer system showing all of the above properties, yet implemented through a human reasoning approach. Terms such as knowledge, belief, intention, and obligation [Min85a][Rao95a] are commonly used in AI. Moreover, some AI researchers also consider emotional agents [Bat94a].

Programming Agent Systems

A number of development tools exist for the design and realization of agent systems, which provide the developer with a model of an agent and some agent prototypes based on the considered model. Easy modeling and programming of a multi agent system is then possible by composing some of the pre-defined agent models. Every complete environment also offers a reliable communication infrastructure enabling interoperability among agents which can rely on the well known agent communication language KQML or can be realized according to some international standards (i.e. FIPA). Actually a huge number of such tools can be found; many of them are distributed with an open-source licensing policy and for the other ones evaluation versions can easily be obtained through the relevant web-sites download areas. Depending on many factors, despite they show similar functionality, the literature offers many different naming for such products which have been, in different times, classified as agent platforms as well as agent construction kit or agent systems building toolkit as well as agent infrastructure or environment. The authors are convinced that talking about an Agent Programming Framework is the best appropriate decision and that describing the whole infrastructure as a framework can clearly express the underlying concepts. Therefore, in this paper, everything will be related to an agent framework. After a preliminary evaluation of the available tools the feeling is that it will be hard to get a highly efficient and really re-usable agent system due to the fact that each platform offers a few specific agent architectures which uniquely can be exploited to build up every other agent in the system. A typical example is the case in which a very trivial agent is needed in the system, and it has to be created on a very complex skeleton while a very simple task has to be performed and a very light structure would be sufficient and appreciated. Moreover, custom applications cannot guarantee an adequate level of reliability, security, performance, and scalability which are naturally expected to be fundamental features of any distributed enterprise application. Consider, as an example, the case in which an agent-based solution has to be developed in a very typical industrial automation scenario that is decision making in a control system. Such an application must show high availability since the efficiency of a plant must be guaranteed twenty-four hours a day, seven days a week. Standard solutions provide redundancy, both hardware and software, thus the considered application must make available both a replication mechanism and full synchronization functionality. In the event of a crash, the state of the system must be preserved as well as the configuration and state of every agent involved. Transparency to the user is necessary in making all of these tasks. From the point of view of performance, in terms of the execution time for some operations, the redundancy of the architecture can be profitably exploited through a load balancing service for distributed computing. For the sake of security of data in the system, transactional operations should always be executed. Last but not least administration issues must be considered too; a fully integrated agent system will interact with system services and with other operational environments or software solutions. An authentication mechanism can provide the agent system with different roles for different agents with different responsibilities and rights. From the above example, the main features that an agent system should offer appear to be reliability and a deep integration with the operating system services in such a way that a high standard quality level of performance can be guaranteed. At the actual stage, existing agent frameworks do not cover all of the above topics and even some of them do not consider any one of those. Such a lack is depending on a technological matter that is the fact that the evaluated agent platforms are open source products whose first aim is that of being portable on any operating systems as well as multiple hardware configurations. On the contrary, the scenario which has been introduced in the above example described a system strongly dependent on the operating environment, and fully exploiting its advanced functionality in terms of services. By considering and comparing available tools and technology devoted to the realization of distributed enterprise applications, it immediately emerges that platform independency leads to an abstraction from the underlying operating system which can only be accepted when services are implemented within the application itself, which results in more than duplicating development time and cost. In a few words, the more complex the application aims to be, the higher the level of integration has to be.

In this work portability issues are only marginally addressed (see Section 4.3) and a primal decision has been taken enabling the realization of an agent programming framework for non-trivial applications rather than a portable one. The adopted solution is that of working exclusively in a Microsoft .NET environment due to some observations done on possible exploitation of the novel concept of Application Domain introduced with the .NET framework (Microsoft Corp.). A more detailed description of such feature in the .NET framework follows in Section 3. An alternative solution to the proposed one can be the use of the J2EE (Java2 Enterprise Edition) platform. The J2EE platform also can offer a wide variety of services and integration capability but proprietary solutions are to be implemented in this case toot as soon as the needed degree of integration grows over a minimum relevant to a reduced set of services which can be used on any platform.

3. THE .NET FRAMEWORK

The Microsoft .NET framework (just .NET in the following) can be considered a support infrastructure enabling the development of distributed applications [Pla01a]. It is also commonly defined as a component-oriented programming platform [Min01a].

The .NET Architecture and CLI

The core of the .NET platform is the so called CLI (Common Language Infrastructure). Any .NET application can be run on any operating system or hardware platform where the CLI is implemented. Above the CLI a rich collection of libraries is implemented and they will be referred to as frameworks. Such libraries provide application programming interfaces or programming abstractions for a wide range of application development tasks. The adoption of .NET also enables language interoperability: components developed by using different languages can interoperate as long as they conform to some rules provided by the Common Language Specification (CLS). Apart these general considerations relevant to the framework, the Common Language Infrastructure will be examined more in details. At the heart of the CLI lies the Common Type System (CTS) which supports all the types and associated operations expected in modern programming languages. This unique set of type definitions is enforced across all languages targeting .NET; this uniform view of primitives, enables crosslanguage interoperability. Source code is compiled into an intermediate format consisting of Common Intermediate Language (CIL) and metadata. In contrast to the COM and CORBA models where programmers were forced to explicitly generate component interfaces and they had to do that by using complex APIs and Interface Definition Languages, in .NET the metadata is automatically generated and persisted in a language-independent format. The final building block of the CLI is the Virtual Execution System (VES) implementing and enforcing the previously discussed CTS. Metadata provides a standard description of every component available in the run-time, and is thus a common interchange mechanism between application programs, system tools, and the run-time itself. This is made possible because every type defined and compiled in the CLI is based on this CTS; the execution engine can verify the type safety of every data element presented for layout in memory and every piece of code to be executed. For compiling and executing tasks, the source code is translated by front-end compilers into a mix of both Common Intermediate Language (CIL) and metadata. The generated intermediate code carries with it all the information necessary for it to be compiled into native code. Run-time byte-code interpretation is not directly supported by the CLI. The CIL is usually verified and compiled just-in-time (JIT-compiler), method by method, by back-end compilers. Then applications are packaged and deployed in assemblies, a collection of files containing types as well as resources. Both assemblies and their enclosed types are self-describing, and are deployed simply by placing them in a specified application directory.

The Application Domain

The application domain is a particular construct in the CLR that is the unit of isolation for an application. The isolation guarantees the following:

- An application can be independently stopped
- An application cannot directly access code or resources in another application
- A fault in an application cannot affect other applications.
- Configuration information is scoped by application. This means that an application controls the location from which code is loaded and the version of the code that is loaded.

With respect to a process, an application domain is lighter. Application domains are appropriate for scenarios that require isolation without the heavy cost associated with running an application within a process. A process runs exactly one application. In contrast the CLR allows multiple applications to be run in a single process by loading them into separate application domains. Additionally the CLR verifies that the user code in an application domain is type safe.

The C# Language

A short introduction to the language used along all of the presented work is presented, focusing on novelties introduced by such a language with respect to other object-oriented programming languages (Java and C++). Common Language Infrastructure and C# provide the developer with tools and facilities [Mic01a] enabling new functionality in the presented agent software. C# is introduced as a simple, object-oriented, and type-safe programming language clearly derived from C and C++. It is promoted as the natural choice for programming in the .NET platform which includes a common execution engine and a rich class library. In the platform a Common Language Subset (CLS) is defined in such a way that, even using C# which is a new language, complete access to the rich class library already defined with other languages in many past years of development activity is guaranteed. C# itself does not include a private class library. It is claimed that C# encourages good object-oriented programming as well as design practices. C# is similar to Java and to C++ in many concepts and solutions, yet it is different in some fundamental ways and behaviors [Wil02a]. For example, the C# language does not permit functions to exist outside of class declarations, which is, on the contrary, allowed in C++. It has a unified type system, and it utilizes garbage collection to free the programmer from the low-level details of memory management. As with all object-oriented languages, C# obviously promotes re-use through classes.

4. SOFTWARE ARCHITECTURE

Most available development tools offer facilities for the implementation of software agents and for communication among agents within the system, yet they do not implement software architecture able to cope with common problems of synchronization and parallel computing in such a way that an agent cannot perform concurrent actions and only can have a very low level of interaction with other agents in the systems, should they need to modify the world they have been created in. Another key point of the actual software architecture that most common agents developed with some of the above cited tools show, is the one to one correspondence between operating system threads and agents. The software architecture proposed in this paper is alternative to existing solutions.

The Proposed Agent Model

Based on the above review of agent characteristics, a detailed description of the agent model proposed by the authors follows. The structural model that is being considered in this work corresponds to an agent both autonomous and multi-behavior. For the sake of clarity, it is necessary to immediately define the meanings of the terms autonomous and multi-behavior in this context.

Autonomous means that "computational activity is independent on other agents". The only mean for an agent to interact with other agents is performing a communication through an asynchronous messaging system.

Multi-behavioral refers to "an agent with many tasks to be accomplished, all of them to be concurrently executed".

Agents Software Implementation

From a programmer's point of view, two solutions are possible for the implementation of an agent: processes or threads. In the first case one agent is associated to the execution of one process while, in the second of the two cases, one process is shared among multiple agents, each one of which is one thread or even more. The two different scenarios relevant to these solutions are sketched.

Agent-Process. From the point of view of autonomy, this one appears to be the best solution: every agent (process) is completely independent from any other agents (processes) in the system. Some agent platforms exist based on such a solution. Different programming languages and technologies lead to the implementation of agents like operating system processes (instances of executable programs) or code managed by a suited run-time environment (i.e. Java in a Java Virtual Machine on any operating system). In both cases, coordination and management of agents are only possible at an operating system level.

Agent-thread. Recently threads have overcome processes in multi tasking programming. A thread can be seen as a light process and a process can be thought as a multi-thread application with a common, shared memory. The activity of managing threads within a process (e.g. activate, suspend, …) can be made easier than managing processes in the operating system environment. In this case, agents are not intrinsically independent, since they are able to share some resources in the same process, hence they also could be non-autonomous. It is responsibility of the (good) programmer to prevent agents from accessing shared resources thus maintaining autonomy characteristics.

Depending on some operating system services, and on how they are implemented, some further disadvantages given by this solution arise, in particular when related to security issues. In the Microsoft Windows NT/2000 architecture for example an authentication mechanism is present, at a process level, therefore all the threads in the same process will be known through the same identity and they will have equal privileges and authorizations. In the case in which different agents are to be associated to different users with different capabilities of intervention on the system (and on the operating system facilities), agent profiles and policies have to be managed and this cannot be done exploiting operating system view of security mechanism, yet it has to be explicitly programmed.

Most agent platforms are developed in and rely on an object-oriented programming language (typically Java). Then developers are provided with a basic abstract class (maybe the class `AgentClass`) to be extended in order to define more and more specialized classes that is agents with well defined, diverse functionality. Agents inherit from the base class `AgentClass` the ability to live in the agent platform (facilities and services for elementary operations such as naming, messaging, …) and they can then be specialized by defining a set of personal resources (private data structures) and their behaviors (methods). In this case data structures become the attributes of the class which has been derived from the base one.

In order to implement multi-behavior functionality of the agent in the framework, two different solutions can be adopted:

1. Every agent uses one and only one thread. Behaviors are defined through an extension of a base class overriding an entry point method. Agent executes each entry point method for all active behaviors with a round-robin schedule in its thread, without pre-emption.
2. Every agent uses one thread for each behavior. Behaviors are defined through extensions of a base class overriding an entry point method. An agent creates and starts a new thread for each behavior; the thread executes the entry point of its associated behavior concurrently with other threads.

Solution 1. With the adoption of this first solution, no race-conditions or dead-locks occur since behaviors are in execution one at a time as they are scheduled sequentially. On the other hand, because of the non-preemptive multi-tasking model which has been adopted, agent programmers must avoid using endless loops, and must not perform "long" operations within the entry point method. The programmer should split the behavior action in several sub-activities which only require small computation time. Sometimes this could be a difficult task.

Solution 2. With this second solution, behavior actions can be as long as necessary because behaviors are executed in multi-tasking, under a preemptive scheduling policy. In this case state information should be kept as local as possible in order to keep behaviors as decoupled as possible, and to prevent problems such as race-conditions and dead-locks occurring. When agent programmers use resources shared among several behaviors they must use the synchronization functionality provided by the programming language in order to cope with concurrency problems. In both cases, functionality and performance which can be obtained from an agent system result to be strongly dependent on the skills of the programmers.

Proposed Solution

In the proposed solution a slightly different approach to the definition of the Agent class has been adopted. It follows the thought of an agent as a software entity whose state is defined by a particular set of data. Other data exist in the agent but they do not compose the agent state and they are used only by a particular behavior in order to perform some local activity. The data composing the agent state have to be accessible from every behavior but they are collected in specific data structures and they are protected by the risk of race-conditions and dead-lock.

In the following it will be described how this model of the agent has been implemented, and how the programmers should use the classes they have been provided with by the framework.

In order to be hosted in the platform an agent has to be an instance of the unique class Agent that cannot be extended. Hence, the programmer of a new agent should define a new class, extended from another class called `AgentTemplate` which offers a set of methods and attributes suitable for the definition of private data and behaviors of the agent. Although an object-oriented programming language (C#) has been used, the programmers must not define the agent private data as attributes of the extended `AgentTemplate` class, but they have to use the exposed method which allows adding generic object as item of the agent private data collection.

Once the agent template has been defined, the programmers can deploy the new template in the platform and every time a new agent is required, always based on the same agent template, the platform crates a new instance of the Agent class binding it with the template. The agent class executes the behaviors and manages the agent data defined in the template.

To give more details about the framework class library, an agent template is composed by a set of behavior objects that, of course, defines the various activities that an agent can perform, and a set of knowledge objects that represent groups of private data logically connected. A knowledge object contains a set of Object, the base class in the CTS from which any instance of any class is derived. With such a model, a behavior can define and use as many its own private data as necessary without problems of race-conditions or dead-locks since data are not shared with other behaviors. Every shared among behaviors data have to be a data representing the agent state, so it has to be defined in a knowledge object of the agent template. Items in the knowledge are not directly accessible from others classes since they are stored in a private collection. In order to access an item of a knowledge, a behavior have to ask the knowledge for the particular item using an exposed method and the behavior have to signal to the knowledge the begin and the end of a section of code which use one or more knowledge objects.

With this solution, the programmer has not to care about synchronization problems while accessing data shared among various behaviors. The solution, in fact, imposed the use of the specific method for the access to the knowledge objects and the knowledge class cares about synchronization transparently. Should a programmer try to access a knowledge item without signaling the knowledge the section of the code in which he/she wants to access the item, an exception will be raised.

The programmer can not define a section of code that use a knowledge object nested in another section that use a different knowledge object but he/she can define a section of code that access a list of knowledge objects. With these assumptions the knowledge class can perform synchronization check adopting dead-lock avoidance algorithms and provide a simple but effective mechanism for the safe access to shared data of the agent.

Furthermore, the proposed model allows the platform to perform any type of control on the state of the agent and allows the development of advanced features of the platform. For instance, this idea is exploited in order to provide the platform with a transparent mechanism of persistence for agents. Such a service, knowing and managing the data of each agent perform, check every time an agent state change and save the new agent state in a suitable way that could be made with different solutions as serialization, mapping in relational data base or using object oriented data base.

Another advanced feature of the platform which is achievable because of the proposed agent model, is the functionality of enroll the modification to the state of an agent in a transaction with other actions as message transmission.

System Integration and Services

All of the agents in the proposed agent programming framework, are executed in the so called Application Domains, hence they can exploit all of the discussed characteristics (see Section 3.2) coming with the application domain itself. By means of the adoption of this technological solution, results achieved are the following:

- easy management for life-cycle of the agents,
- agents can have different rights/denials or privileges within the same process.

Thanks to the adoption of the .NET technology for the development of the discussed Agent Programming Framework, a strict interaction is achieved with both the operating system and the run-time environment. Moreover, services can be easily developed in such a way that applications can take advantage of replication mechanism from the operating system (e.g. Network Load Balancing, Component Load Balancing). Transactional operations on data coming from non-agent-based applications (RDBMS) are supported as well as messaging systems, both synchronous and asynchronous (e.g. MSMQ Message queuing) flexible, reliable, and safe. Agents developed within the proposed agent programming framework can be integrated with the Microsoft Windows operating system (Server Family) at the Active Directory level. In such a way, the directory facilitator mechanism that every agent platform is provided with, is the same of all of the others components in the system (agent-system and everything else), thus it can be easily accessed from all the applications which can share information on the agents for their own advantage. Moreover agents are directly connected to the Internet through the standard operating system communication facility. Agent mobility is guaranteed, within the boundaries of the domain administered by the server which is hosting the agent framework, without any additional cost in terms of programming. It comes naturally that integration can be easily achieved towards the vast suite of Microsoft server products, from which agents can take much advantage for development of enterprise applications such as e-commerce, workflow management, and application interoperability.

Portability Issues

The present solution is entirely based on the .NET technology and on the use of the Microsoft Windows platform which offers services and facilities which have revealed to be very useful within an agent framework (e.g. directory facilitator, message queuing, and more). Nevertheless the concept of Application Domain is also available in BeOS and Linux hence portability to other operating systems can be foreseen in a next future. Moreover the .NET framework is open to a wide variety of languages and developers will not be tied to the use of the C# language (Microsoft Corp.) if they do not like it. Corel is implementing the .NET CLI as shared source for FreeBSD enabling .NET to move to other platforms and open source implementations are underway in a couple of research projects.

5. CONCLUSIONS

As result, a novel architecture has been presented for software agents development in an agent based system that is an Agent Programming Framework. The work has demonstrated good quality of .NET programming platform applied to agent technology. In particular, using the Application Domains coupled with the language C# has made it possible to have an efficient, reliable, and easy to program agent framework for the development of enterprise agent-based applications. Moreover the integration of agent technology with other systems or the re-use of existing software (e.g. decision algorithm, control techniques) is made possible thanks to the definition of the Agent class has been adopted. An agent-oriented programming paradigm has also been introduced within this agent programming framework.

6. REFERENCES

[App02a] Review of software products for Multi-Agent Systems by Applied Intelligence (UK) Ltd. on behalf of AgentLink, the European Network of Excellence for Agent-Based Computing (IST-1999-29003), June 2002.

[Bat94a] Bates, J., “The Role of Emotion in Believable Characters,” Communications of the ACM 37 (7), pp. 122-125, 1994.

[Mic01a] “Microsoft C# Language Specification”, Microsoft Press, Redmond, Washington, 2001.

[Min85a] Minsky, M. (ed.), The Society of Mind, New York: Simon & Schuster, 1985.

[Min01a] Mingins, C. & Nicoloudis, N., “.NET: a new Component-oriented Programming Platform, Journal of Object-oriented Programming, October/November, 48-51, 2001.

[Nwa96a] Nwana, H.S., “Software Agents: an Overview”, Knowledge Engineering Review, Vol. 11, No 3,1-40, September. Cambridge University Press, 1996

[Nwa96b] Nwana, H.S. & Wooldridge, M., “Software Agent Technologies,” British Telecommunications Technology Journal 14 (4), pp. 16-27, October 1996.

[Par97a] Parunak, H.Van Dyke, “Go to the Ant: Engineering principles from Natural MultiAgent Systems,” Annals of Operations Research 75 (1997) 69-101 (Special Issue on Artificial Intelligence and Management Science).

[Pla01a] Platt, D.S., “Introducing Microsoft .NET”, Microsoft Press, Redmond, Washington, 2001.

[Rao95a] Rao, A. S. & Georgeff, M. P., “BDI Agents: From Theory to Practice,” In Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS-95), San Francisco, USA, June, pp. 312-319, 1995.

[Wil02a] Williams, M., “Microsoft Visual C# .NET”, Microsoft Press, Redmond, Washington, 2002.

[Woo95a] Wooldridge, M., Jennings, N.R., “Intelligent agents: Theory and practice”, The Knowledge Engineering Review 10, 2,115-152, 1995.

[Woo99a] Wooldridge, M., “Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence”, G. Weiss (Ed.), MIT Press, Cambridge, MA, 1999.