An Agent Oriented Programming Language Targeting the Microsoft Common Language Runtime

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

ABSTRACT
In the last decades, a significant growth of agent oriented systems has been observed, which has stimulated a more precise formalism for the definition of both agent and multi-agent systems, as well as the release of a huge number of agent development environment. In this work a new programming paradigm is proposed, that is agent oriented programming instead of object oriented programming. The guidelines for the realization of a suited agent programming language, that is an agent oriented language, can be derived according to the basic characteristics that software agents must have. As well as object oriented programming fully exploits the structured programming basic concepts, agent oriented programming will strongly benefit of both the object oriented model and logic programming theoretical basis. A project for the development of a novel architecture has been presented for software agents' development in an agent based system that is an Agent Programming Framework based on the Microsoft Common Language Runtime (CLR).

Keywords
“Agent Programming Language”, “Microsoft CLR”, ".NET Reflection".

1. INTRODUCTION
Agent and multi-agent systems are a widespread research topic. Agent based technology has been proposed as an effective solution to common requirements of information systems applied to manufacturing, communication, workflow management and many other subjects [Oli99a]. In this paper, the reader is assumed to be familiar with the basic concepts of software agents [Woo95a] and multi-agent systems [Woo99a].

Since the introduction of agent technologies, a large number of agent framework have been proposed. Most of them are based on the object oriented languages and are a class library offering facilities for the common agents' tasks i.e. agent life cycle management, message communication and directory service.

Up to now, the activity of programming agents has been writing code in an object oriented language (typically Java) taking into account that the final result should be an agent rather then a simple piece of code or a process. In such a way an agent can be trivially considered as a set of classes or libraries to be used in an agent based solution. The novel concept that the authors are going to promote is a new programming paradigm that is an agent oriented programming technique. Agent-thinking instead of Object-thinking can help the programmer to better understand agents and their philosophy so that he/she can design and implement the best abilities that each agent should have to better operate.

The proposed work is part of a project aiming to develop an Agent Programming Environment and its complete integration within the Microsoft .NET framework.

The design and development of the Agent Programming Framework were carried on based on the Common Language Runtime (CLR) and the C# language [Wil02a], hence on the .NET Platform [Pla01a]. Adopting .NET as the platform on which develop the agent framework, the authors aim to make the framework able to benefit from the numerous services offered by the platform.

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

The Framework was built-up with the following components:

1. Templates for agents programming
2. Interaction modeler for cooperation among agents in the system
3. Agent oriented programming language

In particular, in this framework, much attention was dedicated to the software architecture of agent themselves rather then to the architecture of agent systems to be developed. Merely speaking, agents in an object oriented language are just objects. By means of the .NET platform capabilities one step ahead is possible, that is driving programmers to agent-thinking instead of object-thinking.

Despite the framework architecture being agent oriented, it can not make the programmer use its software components in the best way. Due to the programmer uses an object oriented language in order to access the framework library (any language targeting the .NET platform), non agent oriented patterns are still possible. So, some common mistakes related with concurrent and distributed computation are possible.

In order to avoid such a disadvantage a specialized programming language is necessary. An agent oriented language with its related compiler can make the programmer access the framework in the correct way and signal every mistake at compiling time.

The third part of the project, that is this paper topic, aimed to develop a compiler of a new agent oriented language derived from the C# grammar, targeting the agent framework class library and hence the CLR. The proposed language offers to the programmer the possibility of writing his/her agent oriented code with a strong agent architecture and avoids common difficulties arising in the programming agent with other tools.

Different approaches to the development of a compiler targeting the CLR and the solution adopted within this work are fully detailed.

2. AGENT PROGRAMMING LANGUAGES

In the last decades, a significant growth of agent oriented systems has been observed, which has stimulated a more precise formalism for the definition of both agent and multi-agent systems, as well as the release of a huge number of agent development environment. The major interest for such discipline came from Artificial Intelligence and Software Engineering which have both given complementary contribution to the creation and the evolution of a new agent oriented programming paradigm. From traditional logic programming, a methodology for specification of both the structure and the behavior of software agents can be derived. Moreover, in the meanwhile, agent platforms and development environments have become very common tools, easily available and usable.

By means of agent-based techniques, problems can be solved and solutions implemented; as long as they can be represented as the interaction of a number of agents, then a suited multi-agent system can be designed, representing that particular problem. Existing development tools for multi-agent systems are entirely based on object oriented programming then an agent becomes an object which can be, sometimes, reductive [Cha02a]. The development of agent systems by means of existing tools is possible yet not so easy! In a few words related problems can be listed as follows:

- Agent platforms based on a specific set of classes that implement agents and their behaviors allow designing agent systems only based on those classes and must be programmed by adding specific libraries in a specific object oriented language (Java in most cases). They provide an agent environment and an infrastructure for communication then it is up to the programmer to keep in mind that an agent system is being realized, and hence to write code according to the definition of agent (e.g. autonomous, proactive, and collaborative).
- Agent models based on the classical AI logic programming (e.g., FOPC, multimodal logic) [Par01a] can easily express the agents' behavior and the interaction model among agents, yet are not supported by development platform in such a way that integration and interaction with other systems can become very hard.

3. AN AGENT FRAMEWORK BASED ON THE CLI

Common features offered by agent-programming framework are a support for the concurrent programming, emerging from the multi-behaviors characteristic of the agent paradigm, a communication layer providing a messaging service for the agents, and a platform able to host the active agents and to manage them.

Basic functions of an agent-platform are standardized by FIPA (Foundation for Intelligent Physical Agents) [FIPA, http://www.fipa.org]. In particular, FIPA provides specifications about agent life cycle management, communication protocols and directory facilities to locate agents on the network.

Many agent-programming frameworks are available as commercial or academic products and most of them are based on the Java platform as well as the scientific community is strongly Java-oriented. For a complete survey of agent frameworks and tools see [Age02a].

Even if Java could be an excellent platform on which develop an agent framework, adopting the Java technology make the integration of the agent based applications with other technologies software difficult. In particular, the authors felt the necessity of an agent framework able to cope easily with application targeting the Microsoft Windows operating system, without using the Java platform. So, one year ago a new project was launched whose aim is developing a novel agent-programming framework entirely based on the .NET platform. The result is a framework with all the necessary classes to help the programmer in developing his/her agents, behaviors, and messages plus a Windows Service acting as agent platform. All the software is written in C# language. Agents can use any .NET class and any .NET program can access the agent-platform and use the hosted agents. Thanks to the adoption of the .NET technology for the development of the agent programming framework, a strict interaction is possible with both the operating system and the run-time environment; integration can be easily achieved towards the vast suite of Microsoft server products, from which agents can take advantage for development of enterprise applications such as e-commerce, workflow management, and application interoperability.

Agent Framework Software Architecture

Main difficulties in design an agent framework concern with the necessity to give the agents autonomous activity and make the agent software component protected by other software entities. Moreover agent should be characterized by multi-behaviors activity. That means concurrent programming issues arise among various agents and within each single agent.

In the proposed framework each agent is activated in a separate Application Domain. Thanks to the Application Domain feature provided by the CLI, agents can have different rights/denials or privileges within the same processes and an easy management for life-cycle of the agent has been achieved.

In order to make the agent able to act with many behaviors a novel solution has been adopted. It aims to avoid the necessity of using synchronization primitives of the adopted object oriented language whereas guarantee the correct functionality of the software architecture.

The proposed solution is based on an agent model that is characterized by two separated set of data. Data that specify the agent state and should be accessible from every behavior are collected in a particular software component of the agent called *knowledge*. Other data that do not compose the agent state and are used only by a particular behavior in order to perform some local activity, are bounded in the *behavior* software component.

The knowledge component lets the behavior components access the data it contains in a concurrent way, but it performs all the necessary tasks to protect the data from the risk of race-conditions and dead-lock.

The solution, in fact, imposes the use of the specific methods for the access to the knowledge objects and the knowledge cares about synchronization transparently. Should a programmer try to access a knowledge item without using properly the specialized methods, an exception will be raised.

The disadvantage of this solution is that each item contained in the knowledge objects is accessed in a late binding mode. That means the C# compiler cannot do the necessary checks to avoid a variety of errors as wrong typed names, implicit type cast or missing methods or parameters calls for every instruction that enroll object of the agent knowledge. Moreover the programmer have to explicitly cast the objects accessed from the knowledge in the right type each time they are used, and the possibly errors in using the methods which allow the access to the knowledge objects are discovered at run-time instead of compilation time. Obviously, the framework does every necessary check and raise an exception for each error should occur but no compiling warnings and errors are possible at this stage.

The project of a new agent oriented language aims to avoid such disadvantage. Its objective is defining a grammar for the definition of an agent template and give the programmer the possibility of define his/her agent with a specialized agent oriented language. A translator, targeting the programming framework and hence the CLI, will use the framework classes in order to create the necessary objects to implements an agent defined with such a language. Obviously, the translator will be able to perform any necessary check during the compilation phase and will provide the possibility of using strong typed object instead of a late binding access.

4. THE PROPOSED AGENT PROGRAMMING LANGUAGE

In a programming language that support a certain paradigm, the key-abstractions of the latter are normally translated in native constructs of the former. This mapping leverages the work of the programmer since it permits the modeling of the solution directly using at code level the elements of the paradigm.

There is no effort in casting the solution into the elements offered by programming language.

A clear example are the object oriented languages: the starting point in creating a program are objects and classes as in the object oriented model the programmer has objects and classes to structure a solution. Since object oriented programming is now of common use, the benefits of this mapping against previous programming paradigms do not immediately appear. This is also the main design objective for the proposed agent oriented programming language: the language will offer basic building block to develop applications agent constructs, that will be modeled adding them behaviors and data for its internal state.

Such a language should simplify the implementation of the basic characteristics that software agents must have. Software agents must be:

- Autonomous
- Strongly oriented to social activity
- Cooperative with other agents to reach common objectives
- Able to share common resources

Following these guidelines the new language main features should be:

- A structure that represent an agent;
- A structure for the knowledge of the agent;
- A structure that models the behavior;

These are all high-level structures, and the agent is the starting point for the creation of a programming module. Like all programming languages the agent oriented one should offer the most common features as expressions and control structures. Expressions and control structures will model the core agent's computation inside behaviors elements, so it is necessary to have high expressiveness. The grammar for expressions and control structures recalls the grammar for object oriented languages, with slight modifications that permit access control to shared objects.

There is another reason to introduce some elements of the object oriented model: agent programming normally deals with complexity, and until now the most powerful way to express and organize complex structures has been the object oriented model. So agent programming will still be based on object oriented programming, even better, it can be considered an extension of object oriented paradigm. As well as object oriented programming fully exploits the structured programming basic concepts, agent oriented programming will strongly benefit of both the object oriented model and logic programming theoretical basis.

A former step in this way has been already done by some researchers [Bus99a] that extended the Java language with some new keywords and structures to model agents. The solution will consider the C# language, will relay on the agent programming framework previously discussed to implement the specific features of agency. Only the previously discussed elements will be added to the C# grammar, and some modifications, as said before, will be necessary to enforce the agent programming model. The key point is that, even if all the structures introduced will have a counterpart in a legal C# program (and this mapping will be automatic), what sees the programmer as basic building blocks are agents, and behaviors. The programmer is obliged to cast the solutions in terms of these blocks, and the authors think that where the agent model is suited and useful this is an advantage rather then a lack.

5. WHY C#

C# is a simple, modern, object oriented, and type safe programming language derived from C and C++. It is the main language in the .NET framework, and combines the raw power of C++ with the high productivity of Visual Basic. Like Java, C# supports completely the entire object oriented paradigm and integrates some useful features that leverage programming at language level such as events and indexers. C# is also the language that mostly embodies the different possibilities of the IL, among the ones offered with the .NET platform. It is a very powerful language by which can be implemented a wide range of applications.

C# has been considered as reference point during the project of the agent programming language not only for its support of the object oriented paradigm, but also for the following reasons:

- Seamless integration capabilities with the Microsoft Windows and .NET platforms
- C# is now an ECMA standard, and that restricts the changes that can be made to its grammar and semantics, differently Visual Basic is not a subject of standard
- Similarity with C, C++ and Java: C# has a grammar that is really similar to the mentioned languages. There are a lot of Java and C++ programmers that can easily take confidence with C#.

All these aspects have made C# the basis for the agent oriented programming language.

6. IMPLEMENTATION ISSUES

The tasks to be achieved are the following:

- The definition of a new grammar, extending the C# language, providing the programmer with new keywords and new programming constraints. Such constraints will drive the programmer to an agent-view of the problems to be solved, allowing only agent oriented actions to be implemented and programmed.
- The development of a parser able to process the newly defined language, thus giving all of the information relevant the generation of the equivalent C# code to be "used" in the agent programming platform.
- The development of a semantic analyzer that performs all the additional semantic checks such as internal state data access, proper synchronization.
- A code generator that produces IL assembly for the Common Language Runtime.

In a few words, a compiler is needed. Relaying on the common organization of today's compiler, the huge task of compiler construction can be leveraged. Normally a compiler is built by two components that are [Aho86a] :

- Front-end: including modules for scanning, lexical analysis, and parsing, its outcome is an intermediate representation of the program, subsequently used for target code generation.
- Back-end: operating on intermediate code, eventually optimizing it, its outcome is machine code customized to the target architecture.

Since the target code is known and it is the intermediate language, it is not necessary to create from scratch all the entire compiler, but just the front-end has to be designed accordingly to characteristics of the new language. Front-end, in fact relays mostly on syntactic and semantic aspects of a programming language that are the means by which a paradigm is supported.

Different solutions have been evaluated:

- Lex [Les75a] and yacc [Joh75a] : these are the most used tools to build lexical analyzers, and parsers. Many other tools extend the original functionalities provided with this programs.
- CSTools [http://cis.paisley.ac.uk/crow-ci0/]: compiler writing tools in the tradition of lex and yacc, but using C# as an implementation language. The tools are written using object oriented techniques that are natural to C# and are provided in source form to assist an understanding of the standard algorithms used.
- Microsoft Code Document Object Model (CodeDOM) : composed by the two namespaces System.CodeDom, System.CodeDom.Compiler, provided inside the Framework Class Library. These libraries provide a useful support in creating compilers for small languages
- Possibly joining the VSIP which would strongly leverage the full integration of the new language within the Microsoft .NET platform, and provide facilities for language implementation and advanced features management.

A first implementation of the main features of the parser has been realized using the Jay parser generator [http://www.cs.rit.edu/~ats/], to build an abstract syntax tree of the program using CodeDOM. Despite the easy and rapid development obtained using this solution, the lacks of CodeDOM in representing all the elements of a program, for further analysis, has convinced the authors to change the implementation model. Assuming C# as reference, it is possible to note that CodeDOM does not cover many elements of its grammar as postfix operators, and other features. These lacks are supplied providing a set of CodeSnippet elements that can represent expressions or statements that are not covered by native classes of CodeDOM. Clearly this solution works in embedding automatic code generation features inside an application, but turns out to be not powerful enough in performing exhaustive program analysis, that occurs during semantic analysis. Another pitfall of CodeDOM is that it binds the "compiler" to a specific language, in the project case to C#, so it is necessary to have a C# compiler to create the IL for the run-time. The new implementation model will provide a complete representation of all the elements of the grammar and will exploit the facilities of .NET framework as Reflection.Emit. A final note on implementation should be done. Since the project is based on the .NET framework it is possible to fully exploit its features to leverage the front-end construction task. The explicit construction of an intermediate representation can be avoided using the code generator classes. A smart solution could be the creation of a full featured abstract syntax tree and then use code generation facilities to create the assembly. In this case only the scanner, the parser and the different tree-walkers that perform semantic analysis have to be implemented.

7. CONCLUSIONS

A project for the development of a novel architecture has been presented for software agents' development in an agent based system that is an Agent Programming Framework. The work has up to now demonstrated good quality of .NET programming platform applied to agent technology and is looking forward to developing a new agent oriented language targeted to the IL. An agent oriented programming paradigm has also been introduced within this agent programming framework. Different possible architectures for the design and development of a compiler targeting the new agent oriented language to the CLR have been exposed.

8. REFERENCES

[Age02a] 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.

[Aho86a] A.V. Aho, R. Seti, J. D. Ullman, Compilers: Principles, Technicques and Tools, Addison Wesley, Reading, Massachussets, 1986.

[Bus99a]. P. Busetta, R. Ronquist, A. Hodgson, A. Lucas, JACK Intelligent Agents - Component for Intelligent Agents in Java, Technical Report 1, Agent Oriented Software Pty. Ltd, Melbourne, Australia, 1999.

[Cha02a] Chaib-draa, B., Dignum, F., “Trends in Agent Communication Language”, Computational Intelligence, vol. 18, n.2, 89-101, May, 2002.

[Joh75a] S. C. Johnson, YACC – Yet Another Compiler Compiler, Computing Science Technical Report 32, AT&T Bell Laboratories, Murray Hill, New Jersey, 1975.

[Les75a]M. E. Lesk, Lex – a lexical analyzer generator, Computing Science Technical Report 39, AT&T Bell Laboratories, Murray Hill, New Jersey, 1975.

[Oli99a] Oliveira E., Fischer K. and Stepankova O.: Multi-agent systems: which research for which applications. Robotics and Autonomous Systems 27, 91-106, 1999.

[Par97a] Parks, D.,“Agent Oriented Programming Languages: a Practical Evaluation”, available on the web at www.cs.berkeley.edu/~davidp/cs263/, December, 1997.

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

[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.