A Discussion of Agent Oriented Programming Languages

R. A. Abbasi
School of Computing and Mathematical Sciences,
Oxford Brookes University,
Gipsy Lane Campus,
Oxford,
OX3 0BP,
UK.
rabbasi@brookes.ac.uk

Abstract: Over the past few years, agent technology has become more and more important in many aspects of computer science. Agents and multiagent systems are being used in a variety of applications and across many fields. It is believed that agent technology will emerge as more powerful, useful and profitable in years to come. In this paper we have discussed the idea of a special purpose language, which could provide a native support, features and environment for the development of agents and multiagent systems, i.e. Agent Oriented Programming Language. We have discussed the need, scope, attributes, historical background, and future of such a language.

1 Introduction

An agent can be defined as a software entity that has certain goals, which it tries to achieve, certain capabilities which it posses, certain knowledge which it has, and is autonomous in respect of its actions. By autonomy we mean that an agent can operate usefully at its own with any intervention from outside world and it has control both of its internal states and behavior.

Agents can perhaps be considered as the next step in a gradual progress of development in the field of software engineering. First of all there was imperative programming where there was a series of instructions executed one after another. Next came the introdution of procedures and the idea of procedural programming. This was quickly followed by the concept of modular programming and with it the idea of data abstraction (Abstract Data Type) and information hiding. This served as the foundation of Object Oriented Programming. In Object Oriented Programming, an object is an encapsulation and abstraction of data with certain methods, which can operate on that data and the state of object can only be altered by the mean of one of its methods. The progress in this field did not stop there and in the last few years, companies like Sun, Borland/Inprise and Microsoft have been putting forward the idea of component based Object Oriented languages as the latest paradigm in programming.
Agent based programming can be considered as a logical extension of this and it can be viewed as a specialized form of an object where no one from outside world can change its state, or as a particular form of a component. In this paper we will consider whether or not there is now a need or use for such an agent based language, and if so what should be it’s characteristics. We are not trying to provide answers to these questions in this paper, merely to stimulate what we feel is much needed discussion.

2 Agent Oriented Programming Language

Until now the Multiagent system have been built either in OO languages like C/C++, and Java or AI languages like Lisp and Prolog. The OO approach currently being the more popular, the basic idea is to define agent as an object and built other feature in that object to make it an agent. There are variety of tools developed both in industry and academia, which can be used to build agent-based system or multi-agent systems.
However if we wish to consider the above evolution of agents and multi-agent systems, one can hypothesize the idea of having an agent oriented programming language. Some work has been done in this context (which will be discussed in section-3), but we feel there is still a lack of programming languages, which can be termed as a true agent oriented.

When we talk about an Agent Oriented Programming Language (AOPL) we must first consider what we mean by the term? What are the features, which a typical AOPL should have? What distinguishes it from a “traditional” OO language? Unfortunately, this question does not have a simple answer, as there is still no real agreement on the definition of an agent. People working in different fields have their own perceptions and understanding of agents hence they have their definitions. For some people intelligence is mandatory attribute of an agent and for other it is autonomy.

Therefore our hypothetical AOPL will have to provide support for a wide range of “key” agent characteristics. We have gathered a list (without giving them a ranking) of what we consider to be the principal “key” characteristics that our hypothetical AOPL will have to support.

**Autonomy for Agents:** Autonomy is one of the distinguishing characteristics of an agent. Nearly all of the people working in the field of agent technology agree on this attribute. Also it is this characteristic which we feel distinguishes an agent from an object in the OOP model. By autonomy we mean that an agent is able to act without intervention of humans and other systems. It has control both of its internal states and behavior. It is directed by a set of tendencies. These tendencies are either individual goals, an agent tries to achieve or survival functions, which an agent attempts to optimize. Any OO language guaranties the encapsulation and information hiding for any object. Similarly any AO language, in our view, should guaranty the autonomy of agents. Hence we believe that any language cannot be termed as a true AOPL until and unless it ensures the autonomy of agents programmed in it.

**Communication:** It is possible for an agent to operate usefully alone without any kind of interaction with other agents but the increasing amount of networking and the distributed nature of both data and application is making such situations rare. The ultimate scenario is a multiagent system, consisting of more than one agent, coordinating, cooperating or even competing with each other. In order to coordinate, cooperate or compete with each other agent have to communicate with each other. Also an agent has to communicate with other entities. These entities can be human users or other systems and software. This communication can be direct or indirect depending on type of approach and architecture being used for the purpose of design and development, but it is hard to think of an agent that does not have to interact or communicate with any other entity. An AOPL should enables programmer of agent to implement communication between agents without any difficulties. It should provide some high level and abstract means to implement different type of communication. Normally agents communicate with each other using an Agent Communication Languages (ACL). There are many standard and definitions of ACL available [8,15,17,19,26,32] and this is still a major area of study and research. Any AOPL should not be bound to a particular ACL, but it should give programmer the freedom of selection of any ACL according to their choice and needs.

**Distribution:** As described above agents normally lives in societies forming multi-agent systems. A typical multi-agent system consists of more than one agents more
likely distributed across the world. Any Ideal AOPL should support this distribution of agent across the world.

**Concurrency:** Multiagent systems are often distributed in their nature. Concurrency is one the important properties of distributed systems. It is difficult to use a programming language for programming and implementing MAS unless it provides concurrency; therefore we argue that an AOPL should provide concurrency.

**Naming:** In multi-agent systems each agent has to have a unique name by which it can be identified uniquely. This unique name is very important for many reasons. Especially when the communication between agents is done it needs a unique name for recipient and sender of message. Also whenever some kind of task is distributed among the agents, it requires a unique name for each of them in order to keep the track of who is doing what. So like any distributed system, the multi-agent systems needs a mechanism to allocate each agent a unique name. Ideally an AOPL should provide some mechanism so that each agent can be named uniquely. This will make life easier for programmers and they would not have to design and implement some naming mechanism, each time they program any multi-agent system.

**Mobility of agent:** A mobile agent is a specific type of agent, which can migrate from one machine to another in a heterogeneous network. Though every agent needs not to be mobile there are some areas of application where they are very useful and some times unavoidable. There are many applications have already been developed using mobile agent and lots of work is being done in this field. Therefore we would argue that a true AOPL should be able to support the mobility of agents as well.

**Representation of mental Attributes:** According to many researchers such as Shoham [29], agents have a mental state. The components of such a mental states are often considered to be belief, obligation and capabilities. Many people would not consider a programming language as an AOPL unless it provided support for representing an agent’s mental state.

**Decision-making:** Whether we consider an agent to have mental state or not it still has to make decisions. This is something, which comes with autonomy. When we say that an agent is able to act without intervention of humans and other systems then we have to build some kind of decision making capabilities in it so that it can decide what to do and when to do. Hence we can’t think of an AOPL with out facilities of implanting decision-making. However what needs to be considered is standard programming language support for conditional expressions sufficient or should we consider incorporating more advanced reasoning mechanisms?

**Knowledge representation:** Related to decision making and the representation of mental state is knowledge representation. An agent can learn by different means. It can learn from experience, it can learn from other agents or even it can learn by perception of its environment. The information which an agent learns need to be represented one way or another. There is a need for some kind of mechanism for knowledge representation, which an AOPL should provide, but care needs to be taken not to restrict the information we can represent by our choice of knowledge representation.

**A successor to OO:** As we discussed before agent orientation can be considered as a progressive development of object orientation. Object Oriented models and techniques are well understood now and are widely practiced. Also many of the current agent based applications are developed and implemented in OO languages like Java and C++. It is
therefore not unreasonable to assume that an AOPL, which is similar, is in its design to OOL will have more chances of acceptance as opposed to any other design.

3 Some Existing AO Models

Now when we have discussed what we mean by an AOPL and what are some of the desirable features of an AOPL it is worth having a look at some existing programming models, which are claimed to be agent oriented. In fact there are many languages developed, mainly in academia, for the development of agent based systems, but here we have selected only five of those for discussion.

3.1 Agent-0

The term AOP was first used by Yaov Shoham in 1989[29]. He presented a new programming paradigm, which is a computational framework that draws from AI, Speech Act theory and OOP. According to Shoham, an AOP system consists of three primary components: a formal language for describing mental state of agents, an interpreted programming language to define and program agents and an agentifier to convert neutral devices into programmable agents. Based on his theory of AOP his proposed a language Agent-0[28,29,30].

Agent-0 follows a simple control loop when executing a program, which at each time step does following tasks.

1) gather incoming messages and update the mental state accordingly
2) execute commitments (using capabilities)

A program in Agent-0 is made up of rules for entering commitment, rules for executing capabilities and a set of initial beliefs. In Agent-0, commitments can be made only to execute primitive actions; any activity that would require planning must be committed to as the anticipated elements of its plan.

Agents can perform two classes of actions: private and communicative. Private actions are handled by a separate entity (something other than the AOP interpreter) when they are encountered. No mechanism is described in Agent-0 that how private actions will be performed other than notifying the user that they are to be executed. Communicative actions use the speech-act commands to converse with other agents. The four classes of messages in Agent-0 are INFORM, REQUEST, UNREQUEST, and REFRAIN. An INFORM action sends the fact to the receiver agent, a REQUEST notifies receiver agent that the requester would like the action to be realized. An UNREQUEST is the inverse of a REQUEST. A REFRAIN message asks that an action not be committed to by the receiving agent.

Agent-0 was first effort in the direction of an AOPL. It has many deficiencies. It does not define any mechanism how the private action will be performed also the speech acts supported in communicative action are very few. It does not address the issue of communication of message from one agent to another but just assume that it is responsibility of underlying platform to pass the message from one agent to another.
Further more, in Agent-0 no agent would ever commit to anything unless asked to by another agent.

Though Agent-0 provide some support to the features like mental attributes but it lacks in the provision of other features. So because of above mentioned deficiencies and lack of provision of some key features Agent-0 cannot be used for agent development in current heterogeneous environment and we cannot term it as a true AOPL in modern sense.

3.2 The PLACA Extension of Agent-0

PLACA (PLAning Communication Agents) was proposed by Rebecca Thomas [31] as an extension of Agent-0. In Agent-0, one of the major limitations can be termed plan-directedness, i.e. no commitments to anything but primitive actions can be realized, as the agent has no mechanism for developing goals. PLACA attempts to make the agents capable of making the plan.

Though PLACA is an improved shape of Agent-0 but being a descendant of Agent-0, it inherits all of its deficiencies discussed above. Hence because of same reason as we discussed for agent-0, PLACA does not fit in our criteria of AOPL.

3.3 The Agent-K Extension of Agent-0

Agent-K [6] is an effort to introduce some standards in the message passing functionality of Agent-0. In order to ensure that message written in other languages can be handled, Agent-K integrates the syntax of Agent-0 with the format of KQML. Two major changes in Agent-0 are introduced by Agent-K:

Outgoing INFORM, REQUEST, and UNREQUEST message actions are replaced with one command, KQML that takes as its parameters the message, the time, and the KQML type. Incoming messages, however, are converted into the base Agent-0 types - the three message types are expressive enough to encapsulate the other KQML types almost completely.

Agent-K allows many commitments to match a single message. In Agent-0, this was not defined, and the interpreter simply selected the first rule that matched a message.

Though Agent-K enhances the communication capabilities of agents but still it inherits all other deficiencies of agent-0. So again we cannot describe it as an ideal AOP language.

3.4 Mozart

The language Mozart, developed by the Mozart consortium [23], is a general-purpose development platform. It was designed specially to support concurrency, distribution, resource-aware computation, and symbolic computation and inferencing. It implements Oz, a concurrent object oriented functional language which has a simple underlying model based on concurrent constraints extended with state and higher-orderness. Oz combines concurrent and distributed programming with logical
constraint-based inference. This feature makes it a good choice for multi-agent System development.

Mozart does support some of the features we have listed above like concurrency, distribution and some form of support for knowledge representation and decision-making. But it does not provide any support to some other key features like autonomy and mobility. Also it is based on another language which is basically an object oriented in nature. Agents based application can be developed using this language but this can be done in many other languages (C/C++, Java, Prolog etc) but this does not make it an Agent Oriented Language.

3.5 APRIL

Another language called APRIL (Agent Process Interaction Language) was developed at Imperial College London [2,20]. Many of the features in April are taken from other languages; in particular much is owed to Parlog, Erlang, PCN, CSP, Dijkstra's guarded commands, LISP, Prolog and APL. It is claimed to be a distributed symbolic language. The key idea behind April is symbolic based computation. The building blocks in April programs are processes. All processes have globally unique identifier. It provides certain data structuring and expression handling features in order to enable the process to communicate with each other. April higher order features are used for structuring the program and also to provide a facility for migration of code from one process to other.

Though April provides some support for feature like communication, concurrency and to some extent mobility but it cannot be said as true agent oriented language and does not provide support for other features. This could perhaps be best consider to be a Process Oriented Language or an object base concurrent language where active objects are processes.

4 Discussion

We have seen what are the possible attributes of an AOPL. We have also briefly discussed some of existing languages and have evaluated them on our criteria of an ideal AOPL. Apart from the theoretical discussion on these languages, none of these languages are, to our knowledge, widely used for the design and development of agent based systems. Most programmers still prefer to use either existing languages like C++ and Java or some agent builder tools and platform, which are also mostly developed in C++ or Java.

It could be argued that this is because of the so-called old “resistance to change”. But saying this is not enough. We believe there are many reasons for this trend and here we will like to explore some of them.

Agent technology is the subject-area for people working in more than one field with the totally different domains of applications and areas of interest; each group has its own understanding and visualization of the agent. The result is not unexpected there is still no agreement on the definition of entity agent itself. Because of similar reason researchers in the field of agents are still unable to agree on architecture of agent, agent
oriented paradigm, and architecture of MAS. In such an environment each group of
people have their own set of criteria for so-called AOPL. All the efforts made in the
direction of design or proposal of a programming language for agent development is
influenced by the philosophy and theory of one school of though or another. Hence in
this scenario there is a very little chance of global acceptance of any standard language
and its use.

This is still an ongoing debate that whether we need AOP language or not [21]. There
is a group of people, which is of the view that this is right time for introduction of a new
AOPL and it will serve a good step in the progress of the field of agent technology.
While others argue differently. They think that introduction of an AOPL at this stage
will freeze the research in the fields of architecture, coordination mechanism and
communication strategies. They suggest that there is a need to wait for further work to
be done, until there is an agreed upon and well-understood standard, before heading
towards an AOPL if at all.

There is another unsettled question, whether an AOP should be an ext ension of OOP,
Logic Programming or rule based programming or something totally different. We have
problems associated with each approach. There is another idea is to have a hybrid
approach.

There are many different types of agents like intelligent agent, s/w agent, cognitive
agent, reactive agent, interface agent, mobile agent and so on. This is still unclear
whether only one programming language can offer support for all types of agent. If
some one is able to design one programming language, which support the development
of all these type of agent then what will be the its complexity level?

Currently OO paradigm is well understood. There are many OO programming
languages used by the programmer worldwide by the people working the field of
computer sciences. Even most of people working in the field of agent technology are
very familiar with these languages and most of them are using these languages for the
development purposes. Any new language will have to compete these languages in the
market. This is a very important question that whether any new language will be able to
replace these languages at least in the field of agent technology and if answer to this
question is yes than how long it will take to happen.

Agent is a relatively a new but novel concep t and agent oriented paradigm is new
software paradigm. Ideally there should be a language, which could offer a native
support to this paradigm. Like us many people feel the need of such a language. But
unfortunately because of the reason discussed above we are not very optimistic about
the success and global acceptability of any new AOPL.

5 Conclusions

AOP is a possible new paradigm and it is different from and extends existing
software development methodologies. No existing language can be termed as a
complete AOPL and we would argue that there is a scope and need for such a language.
We have detailed the list of attributes, which we consider that an ideal AOPL should
posses. However at the same time we consider that, because of reason discussed in
section 0, there is very little chance of global acceptance of any new AOPL right now.
It looks like as people will keep on using different tools for the development of their agent based systems depending on the their need, tastes, areas of interests and level of expertise. Work done by different organizations like FIPA, we hope, will able to introduce some kind of standards of agents and AOP. Once there is an agreed standard then the introduction of new AOPL might get global acceptance.

One of the possibilities which currently seems likely is that we will end up with more than one programming languages each for a specific type of agent e.g. one language for mobile agent, other for interface agent and so on. We should not consider this to be strange. If look at the history of “conventional” programming languages we see that we have had different programming languages from different types of program (like COBOL for business and Fortran for scientific purposes). It is only later that these languages have been combined into more general-purpose languages. Currently we are not able to find any work done in this direction in term of programming language. But this trend is already bean followed in the some of the agent builder tool. Like IBM Aglet [1] is a tool for the development of mobile agent while JADE [13] support development of agents in compliance with FIPA architecture. One can envisage that this trend will travel from the tools towards the language in future giving birth to more than one agent development language each supporting a specific type of agent.

Another desirable approach would be the development of language independent libraries, or at least an interface description for a set standard libraries. Currently we have different tools available for development of agents but nearly all of such tools are based on some specific language. A more sophisticated approach can be to have language independent class libraries for the development on agent based systems, similar to the POSIX library standard for operating systems. This might introduces some kind of standardization in the agent development methodologies, which can be refined and extended into a proper and broad base AOPL later.

Lately Sun Microsystems is reviewing a specification, JSR00087, for Java Agent Services. [16]. The Java Agent Services Project [14] is an initiative to define an industry standard specification and API for the development of network agent and service architecture. We know that Java is alr eady favorite for a big class of programmer working in the filed of agent technology. The introduction of new Java Agent Services will be more than welcomed by a this class. We think that this might put the birth of a new AOPL further on hold.

References

11. IBM (1996): Intelligent Agent Resource Manager, Open Blueprint G325-6592-0