Abstract

The paper is a reflection on agent-based AI. Our contention is that AI research should focus on interactive, autonomous systems, that is, on agents. Emergent technologies demand so. We will see how recent developments in (multi-) agent oriented research have taken us closer to the original AI goal, namely, to build intelligent systems of general competence. Agents are not the panacea though. We will point out several areas such as design description, implementation, reusability and security, that must be developed before agents are "universally" accepted as the AI of the future.

Now that we have entered the new millennium, it is time to review where we are and where we are going to in Artificial Intelligence (AI). Although substantial progress has been made in the development and application of AI technology, much remains to be done. As Nils J. Nilsson has stated (Nilsson, 1995) We have to distinguish between Intelligent programs and the specialised systems, that is, the tools, that they use. No doubt, building the tools is important. But working on the tools alone has not moved us closer to AI's original goal.

The most persistent and troubling of all the criticisms, to paraphrase Michael Wooldridge (Wooldridge, 1998), is that AI has simply failed to deliver on its basic promise, that is, to build intelligent systems of general competence. There is thus a feeling that after more than 50 years of AI research 1, it is time to go back to the Good Old-Fashioned Artificial Intelligence (GOFAI)(Nilsson, 1995).

Times have changed though, and the new GOFAI cannot stick with so-called traditional software. Emergent technologies such as the internet demand personal, continuously running, autonomous systems.

Like many AI authors (see Russell and Norvig, 1995; Huhns and Singh, 1998), we believe that for AI systems to perform successfully they must be able to behave in an autonomous, flexible manner in unpredictable, dynamic, typically social domains. In other words, the new GOFAI should develop agents.

By autonomy, we mean the ability of the systems to make their own decisions and to execute tasks on the designer’s behalf. Delegating some responsibilities to the system and avoiding the tedious task of writing down the corresponding code is certainly very attractive. Moreover, in unknown scenarios where it is difficult to control directly the behaviour of our systems, the ability of acting autonomously is essential (e.g., NASA's RAX space-ship controller (http://rax.arc.nasa.gov)).

It is precisely their autonomy that defines agents. Even though it is possible to encapsulate some behaviors by specifying private methods in object-oriented (OO) programming, the system cannot deny access to a method that has been declared “public”. That means that the object must execute the method anytime it is requested to do so. On the contrary, agents must decide by themselves whether to execute their methods according to their goals (agents must be pro-active), preferences, and beliefs. As it is usually said “what objects do for free, agents do for money” (Wooldridge 1999). Agents typically operate according to a Remote Programming Paradigm, or computer-to-computer programming. Remote programming enables the client (user) to store on the server (host) not only the procedures but also any accompanying instructions and peripheral data. Each time the events specified in the instructions occur, the server calls the procedures and executes on-site the instructions, without any further intervention from the client computer.

Secondly, agents must be flexible. When designing agent systems, it is impossible to foresee all the potential situations an agent may encounter and specify an agent behavior optimally in advance. Agents therefore have to learn from and adapt to their environment. This task is even more complex when nature is not the only source of uncertainty, and the agent is situated in an environment that contains other agents with potentially different capabilities, goals, and beliefs. For example, the components of interaction in the internet (potentially ad-

---

1For Americans, AI started off at MIT in 1957 with Marvin Minsky’s and John McCarthy’s Artificial Intelligence Project. Europeans, on the other hand, cite Alan Turing’s article in Mind (Turing, 1950) as the origin of AI.
versarial agents, protocols, and languages) are not known a priori.

Of course, in restricted domains, there is no need to learn anything. Yet, in such cases there is no need either to design agents. An OO approach would be more appropriate. However, intelligence and learning are closely tied in uncertain domains where autonomous agents must postpone making decisions until relevant information has been acquired. Not surprisingly, learning has recently received increasing attention in connection with agents and multi-agent systems (Weiss, 1997; Sen, 1998, Weiss 1999; Alonso et al., 2001; Kudenko and Alonso, 2001). Alternatively, novel planning algorithms such as Graphplan (Blum and Furst, 1997) have been extended for conditional planning (Anderson et al., 1998), contingent planning (Weld et al., 1998), temporal planning (Smith and Weld, 1999), and probabilistic planning (Blum and Langford, 1999)—in an attempt to handle uncertainty. It is too early to say which planning algorithm will dominate (see AIPS-00 Planning Competition, http://www.cs.toronto.edu/aips2000/).

An agent must also show a social attitude. In an environment populated by heterogeneous entities, agents must have the ability to recognise their opponents, and form groups when it is profitable to do so. Developing agent teams has been a topic of intensive research in the agent community (Grosz and Kraus, 1999; Tambe et al., 1999) but it is not a coincidence that most agent-based platforms (such as AgentBuilder (http://www.agentbuilder.com/), Jack (http://www.agent-software.com.au/), MadKit (http://www.madkit.org/), and Zeus (http://www.labs.bt.com/projects/agents.htm)) incorporate multi-agent tools. Some authors (e.g., Zambonelli et al., 2000) do state that agent-oriented software engineering needs to be developed precisely because there is no first class notion of organisational structure (the very essence of multi-agent systems) in the OO world.

Besides, the new systems must be general. An agent must have the competence to display an action repertoire general enough to preserve its autonomy in dynamic environments. Again, it becomes clear that the ability to learn and adapt is one of, if not the most important feature of intelligence. An agent can hardly be called intelligent if it is not eventually able to perform well when you put it into an environment different from the one it was initially designed for.

Finally, the use of agent technology to build intelligent systems has the following additional advantages:

- Agents are a very natural metaphor to understand and use intelligent systems. It is very intuitive to model our systems as we think of ourselves, that is, as human agents with beliefs, desires, and intentions (a la BDI (Rao and Georgeff, 1991)). Agents thus fit perfectly in the "user-friendly" paradigm. Apart from executing actions on the user’s behalf, agents can also assist users teaching or training them, helping different users collaborate, and monitoring events and procedures.

The production of artificial and virtual pets (e.g., AIBO (http://www.jp.aibo.com), ASIMO (http://world.honda.com/ASIMO/), Creatures (http://www.cyberlife.co.uk)), and the development of affective and believable characters (such as STEVE (Johnson et al., 1998)) are examples of agent-based applications.

More importantly, agents are regarded as the technology in which the IT and telecomms sectors should converge. They have a role to play at the client-side of such systems, providing customers with personalised, pro-active interfaces to new services and products. They have a role to play as middleware, putting users in contact with the goods and services that best suit their needs. And they have a role to play as servers, cooperating and negotiating on behalf of organisations and other end users. Moreover, as Nick R. Jennings put it "Electronic Commerce is the most important application for Agent Technologies, because it is reality-based and constitutes a massive market" (Noriega and Sierra, 1999). Not surprisingly, agents are being extensively used to implement electronic markets and electronic auctions (e.g., Kasbah (Chavez and Maes, 1996), Bazaar (Zeng and Sycara, 1997), MAGMA (Tsvetovatyy, 1997), MAGNET (http://www-users.cs.umn.edu/~gini/magnet.html), FishMarket (http://www.iiia.csic.es/Projects/fishmarket/newindex.html)). Moreover, the Trading Agent Competition (TAC, http://auction2.eecs.umich.edu/) provides a platform to test e-commerce applications.

Generally speaking, the design and implementation of multi-agent systems is an attractive platform for the convergence of various AI technologies. That is the under-
lying philosophy of the RoboCup competitions (for both robots and simulators), where teams of agents must display their individual and collective skills in real-time (Asada et al. 1999). Even though it is hard to find a practical application for artificial soccer other than pure entertainment, the idea of using such a popular spectacle as a demonstration of AI systems has been very successful (http://www.robocup.org).

- When the domain involves a number of distinct problem solving entities (or data sources) which are physically or logically distributed (in terms of their data, expertise or resources), an agent-based approach can often provide an effective solution. The (Computational) Grid Forum (http://www.gridforum.org) is an ideal scenario to prove the suitability of agents to solve inherently distributed problems (http://www.cs.cf.ac.uk/User/O.F.Rana/agent-grid-2002/).

- When the domain is so large, sophisticated, or unpredictable, the overall problem can indeed be partitioned into a number of smaller and simpler components, which are easier to develop, and maintain, and which are specialists at solving the constituent problems. It has been shown that distributing problem solving over several levels of abstraction reduces the time complexity of the planning process (Korf, 1987; Montgomery Durfee, 1993). Moreover, in a hierarchical multi-agent system, the learning module can be applied within each abstract problem space. Since the abstract problem spaces define more general and thus simpler problems and theories, agents learn more efficient and appropriate rules (Knoblock et al., 1991).

No doubt, our defence of agent-based technology as the new revolution in AI might seem out of date. After all, agent research and technology groups have been established by such companies as Microsoft, IBM, Sun Microsystems, AT&T, and Netscape. Besides, agent-based applications have been reported in manufacturing, process control, telecommunications systems, air traffic control, traffic and transportation management, information filtering and gathering, electronic commerce, business process management, entertainment and medical care (Jennings and Wooldridge, 1998). Nonetheless, one of the key problems of recent years has been the divide between theoretical work in agent-based systems and its practical complement which have, to a large extent, developed along different paths. As a consequence, designers lack a systematic methodology for clearly specifying and structuring their applications as multi-agent systems. Most agent-based applications have been designed in an *ad hoc* manner

— either by borrowing a methodology (typically an object-oriented one) and trying to shoe-horn it to the multi-agent context or by working without a methodology and designing the system based on intuition and past experience.

(Jennings et al.,1998)

In particular, the following areas must be developed:

1. **Design description**: Conceptual modelling and specification languages used to describe a system design, that is, a Unified Agent Modelling Language (UAML). For example, there is no universally accepted definition of the term agent. Any control system of an UNIX daemon can be viewed as an agent;  

2. **Properties and analysis**: Methods and techniques to specify, establish, and verify properties of an agent system.

Since the mid 80’s, problems with symbolic reasoning led to so-called reactive agents. As Michael Wooldridge puts it (Wooldridge, 1999), "a major point of such systems is that the overall behaviour emerges from the interaction of the component behaviours when the agent is placed in its environment. There are obvious advantages to reactive approaches: simplicity, economy, computational tractability, robustness against failure. However, such systems make it very hard to engineer agents to fulfill specific tasks. Ultimately, there is no principled methodology for building such agents: One must use a laborious process of experimentation, trial and error to engineer an agent.”.

From a user point of view, the agent must exhibit somewhat predictable behaviour and provide some sort of explanation for its actions. Designers then need a formal semantics to verify formally that a given behaviour conforms to the specification;
3. Implementation: Agent and market-oriented programming languages, as well as standard co-ordination protocols and agent communication languages (ACL).

While some OO features such as abstraction, inheritance and modularity make it easier to manage increasingly more complex systems, JAVA (or its distributed extensions JINI and RMI) and other OO programming languages cannot provide a direct solution to agent development. As for "real" agent-oriented programming, Agent-0 (Shoham, 1993) and the like (PLACA (Thomas, 1993), Agent-K (Davies and Edwards, 1994), Elephant 2000 (McCarthy, 1999)) are still small and simple languages, used mainly to test ideas about agent-oriented programming rather than for the development of any realistic systems.

Another issue to address in a multi-agent scenario is interoperability. The debate should not be focussed exclusively on the pros and cons of different languages and protocols (FIPA (http://www.fipa.org), KQML (http://www.cs.umbc.edu/kqml/), CORBA (http://www.corba.org)), but also on ontologies. Currently, ontologies are often specified informally or implicit in the agent implementation. For true interoperation, agents will need explicitly encoded, sharable ontologies;

4. Reusability: Methods and techniques to specify and maintain reusable models and reusable software for multi-agent systems, agents and agent components. It was made abundantly clear in AgentLink2’s last meeting (Amsterdam, December 2001) that users do not like new expensive difficult-to-learn systems (http://www.agentlink.org). They would rather reuse their old systems and add to them new technology.

Reusability also refers to mobility. Obviously, we want our agents to roam wide area networks such as the WWW. That is, we want to (re)use them continuously in different scenarios. Once again, for this ideal to be realized, standard languages and protocols are needed;

5. Security: For individuals to be comfortable with the idea of delegating tasks to agents, they must first to trust them. Issues that need to be addressed include: authentication, privacy of communication and user’s personal profile information, trust, auditing, accountability and defence against malicious or incompetent agents (see (Castelfranchi et al., 2000) for an overview on “Trust and Agents”).

We have focussed this paper on software agents or softbots. We would like to finish mentioning the work that has been done in robots. The bad news is that (Menzel and D’Aluisio, 2000)

Despite 25 years of intensive research aimed at the development of a robotics science, the statistics of actual robot use in industry have remained essentially constant: more than 90 per cent of industrial robots are use for spot welding and spray painting...

Hopefully, the introduction of qualitative analyses in vision (active or purposive vision (Bianchi and Rillo, 1996)), the use of fuzzy logics for navigation ("Thinking Cap" (Parsons et al., 1999)) and the development of Behavior-based robotics and hybrid architectures (Arkin, 1999), will change this situation in the near future.

REFERENCES


International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology.


**BIOGRAPHICAL STATEMENT**

Dr Eduardo Alonso (http://www.soi.city.ac.uk/~eduardo) is a Lecturer in Computing at City University, London. He is also a member of The Society for the Study of Artificial Intelligence and the Simulation of Behaviour (SSAISB) Committee, the largest Artificial Intelligence Society in the United Kingdom. Dr Alonso’s research has been focused on the development of co-ordination protocols and the implementation of knowledge-based learning techniques in multi-agent scenarios, particularly in e-commerce applications.