Agent Frameworks

Reinier Zwitserloot
Delft University of Technology, The Netherlands

Maja Pantic
Delft University of Technology, The Netherlands

INTRODUCTION

Software agent technology generally is defined as the area that deals with writing software in such a way that it is autonomous. In this definition, the word *autonomous* indicates that the software has the ability to react to changes in its environment in a way that it can continue to perform its intended job. Specifically, changes in its input channels, its output channels, and the changes in or the addition or removal of other agent software should cause the agent to change its own behavior in order to function properly in the new environment. In other words, the term *software agent* refers to the fact that a certain piece of software likely will be able to run more reliably without user intervention in a changing environment compared to similar software designed without the software agent paradigm in mind. This definition is quite broad; for example, an alarm clock that automatically accounts for daylight savings time could be said to be autonomous in this property; a change in its environment (namely, the arrival of daylight savings time) causes the software running the clock to adjust the time it displays to the user by one hour, preserving, in the process, its intended function—displaying the current time. A more detailed description of agent technology is available from Russel and Norvig (2003).

The autonomous nature of software agents makes them the perfect candidate for operating in an environment where the available software continually changes. Generally, this type of technology is referred to as multi-agent systems (MAS). In the case of MAS, the various agents running on the system adapt and account for the other agents available in the system that are relevant to its own operation in some way. For example, MAS-aware agents often are envisioned to have a way of negotiating for the use of a scarce resource with other agents.

An obvious start for developing MAS is to decide on a common set of rules to which each agent will adhere, and on an appropriate communication standard. These requirements force the need for an underlying piece of software called an agent framework. This framework hosts the agents, is responsible for ensuring that the agents keep to the rules that apply to the situation, and streamlines communication between the agents themselves and external sensors and actuators (in essence, input and output, respectively). This paper will go into more detail regarding the advantages of MAS and agent frameworks, the nature and properties of agent frameworks, a selection of frameworks available at the moment, and attempts to draw some conclusions and best practices by analyzing the currently available framework technology.

BACKGROUND: RESEARCH MOTIVATIONS

An agent framework and its use as a base for MAS technology already has been successfully used as the underlying technology for most teams participating in the robot soccer tournament (Tambe, 1998). The robotic soccer tournament requires that all participating robot teams operate entirely under their own control without any intervention by their owners. The general idea of independent autonomous robots working together to perform a common task can be useful in many critical situations. For example, in rescue situations, a swarm of heterogeneous (not the same hardware and/or software) agents controlling various pieces of hardware fitted onto robots potentially can seek out and even rescue

people trapped in a collapsed building. The ideal strived for in this situation is a system whereby a number of locator robots, equipped with a legged transport system to climb across any obstacle and sporting various location equipment such as audio and heat sensors, will rapidly traverse the entirety of the disaster area, creating a picture of potential rescue sites. These, in turn, serve as the basis for heavy tracked robots equipped with digging equipment, which work together with structure scanning robots that help the digging robots decide which pieces to move in order to minimize the chances of accidentally causing a further collapse in an unstable pile of rubble. Equipment breaking down or becoming disabled, for example, due to getting crushed under an avalanche of falling rubble, or falling down in such a way that it can't get up, are not a problem when such a rescue system is designed with MAS concepts in mind; as all agents (each agent powering a single robot in the system) are independent and will adapt to work together with other robots that currently are still able to operate, there is no single source of system failure, which is the case when there is a central computer controlling the system. Another advantage of not needing a central server is the ability to operate underground or in faraway places without a continuous radio link, which can be difficult under the previously mentioned circumstances.

A crucial part of such a redundancy-based system, where there are no single sources of failure, is to have backup sensor equipment. In the case of conflicts between separate sensor readings that should have matched, agents can negotiate among themselves to decide on the action to take to resolve the discrepancy. For example, if a teacup falls to the floor, and the audio sensor is broken, the fact that the video and image processing equipment registered the fall of the teacup will result in a negotiation session. The teacup fell according to the agent controlling video analysis, but the audio analyzer determined that the teacup did not fall—there was no sound of the shattering cup. In these cases, the two agents most likely will conclude the teacup did fall in the end, especially if the audio agent is capable of realizing something may be wrong with its sensors due to the video backup. Or the agents together can determine if further detail is required and ask an agent in control of a small reconnaissance robot to move to the projected site where the teacup fell and inspect the floor for cup fragments. The system will still be able to determine the need to order new teacups, even though the audio sensor that usually determines the need for new teacups currently is broken. This example displays one of the primary research motivations for multi-agent systems and agent frameworks—the ability to continue operation even if parts of the system are damaged or unavailable. This aspect is in sharp contrast to the usual state of affairs in the world of computer science; for example, even changing a single bit in a stream of code of a word processor program usually will break it to the point that it will not function at all.

Another generally less important but still significant motivation for MAS research is the potential benefit of using it as a basis for systems that exhibit emergent behavior. Emergent behavior refers to complex behavior of a system of many agents, even though none of the individual components (agents) has any kind of complex code. Emergent behavior is functionally equivalent to the relatively complex workings of a colony of ants capable of feeding the colony, relocating the hive when needed, and fending off predators, even though a single ant is not endowed at all with any kind of advanced brain function. More specifically, ants always will dispose of dead ants at the point that is farthest away from all colony entrances. A single ant clearly cannot solve this relatively complex geometrical problem; even a human being needs mathematical training before being able to solve such a geometric problem. The ability to find the answer to the problem of finding the farthest point from a set of points is an emergent ability displayed by ant colonies. The goal of emergent behavior research is to create systems that are robust in doing a very complex job, even with very simple equipment, contrasted to products that are clunky to use, hard to maintain, and require expensive equipment, as created by traditional programming styles. Areas where emergent behavior has proven to work can be found first and foremost in nature: Intelligence is evidently an emergent property; a single brain cell is government by extremely simple rules, whereas a brain is the most complex computer system known to humankind. This example also highlights the main problem with emergent behavior research; predicting what, if any, emergent behavior will occur is almost impossible.

Conversely, figuring out why a certain observed emergent behavior occurs, given the rules of the base component, usually is not an easily solved problem. While the neuron is understood, the way a human brain functions is not. Still, research done so far is promising. The most successes in this area are being made by trying to emulate emergent behavior observed in nature. Bourjot (2003) provides an example of this phenomenon. These promising results also are motivating agent framework research in order to improve the speed and abilities of the underlying building blocks of emergent behavior research—simple agents operating in an environment with many such simple agents.

PROPERTIES OF AGENT FRAMEWORKS

Many different philosophies exist regarding the design of an agent framework. As such, standardization attempts such as MASIF, KQML, and FIPA mostly restrict themselves to some very basic principles, unfortunately resulting in the virtual requirement to offer features that exceed the specification of the standard. Possibly, this aspect is the main reason that standards adherence is not common among the various agent frameworks available. Instead, a lot of frameworks appear to be developed with a very specific goal in mind. As can be expected, these frameworks do very well for their specific intended purpose. For example, hive focuses on running large amounts of homogenous (i.e., all agents have the same code) agents as a way to research emergent behavior and is very useful in that aspect. This section analyzes the basic properties of the various agent frameworks that are currently available.

- **Programming Language:** Implementing the agent will require writing code or otherwise instructing the framework on how to run the agent. Hence, one of the first things noted when inspecting an agent framework is which language(s) can be used. A lot of frameworks use Java, using the write-once-run-anywhere philosophy of the language designers to accentuate the adaptable nature of agent software. However, *C++*, Python, and a language specification specialized for creating distributed software called *CORBA* also are available. Some frameworks take a more specific approach and define their own language or present some sort of graphical building tool as a primary method of defining agent behavior (e.g., ZEUS). A few frameworks (e.g., MadKit) even offer a selection of languages. Aside from the particulars of a potential agent author, the programming language can have a marked effect on the operation of the framework. For example, *C++* based frameworks tend not to have the ability to prevent an agent from hogging system resources due to the way natively compiled code (such as that produced by a *C++* compiler) operates. Java programs inherently can be run on many different systems, and, as a result, most Java-based frameworks are largely OS and hardware independent. Frameworks based on CORBA result in a framework that has virtually no control or support for the agent code but is very flexible in regard to programming language. Due to the highly desirable properties of system independence offered by the Java programming language, all frameworks reviewed in the next section will be based on the Java language.
- **State Saving and Mobility:** The combination of the autonomous and multi-agent paradigm results in a significant lowering of the barrier for distributed computing. The agent software is already written to be less particular about the environment in which it is run, opening the door for sending a running agent to another computer. Multi-agent systems themselves also help in realizing distributed computing. An agent about to travel to another system can leave a copy of itself behind to facilitate communication of its actions on the new system back to its place of origin. As a result, a lot of agent frameworks offer the ability to move to another host to its agents (e.g., Fleeble, IBM Aglets, NOMADS, Voyager, Grasshopper). The ability to travel to other hosts is called mobility. Advantages of mobility include the ability of code, which is relatively small, to move to a large volume of data, thus saving significant bandwidth. Another major advantage is the ability to use computer resources (i.e., memory, CPU) that

are not otherwise being used on another computer—the creation of a virtual mega computer by way of combining the resources of many ordinary desktop machines. Inherent in the ability to move agents is the ability to save the state of an agent. This action freezesthe agent and stores all relevant information (the state). This stored state then either can be restored at a later time or, alternatively, can be sent to another computer to let it resume running the agent (mobility). The difficulty in true mobility lies in the fact that it is usually very difficult to just interrupt a program while it is processing. For example, if an agent is accessing a file on disk while it is moved, the agent loses access to the file in the middle of an operation. Demanding from the agent framework that it check in with the framework often, in a state where it is not accessing any local resources that cannot be moved along with the agent, generally solves this problem (Tryllian).

- **Communication Strategy:** There are various communication strategies used by frameworks to let agents talk to each other and to sensors and actuators. A common but hard-to-scale method is the so-called multicast strategy, which basically connects all agents on the system to all other agents. In the multicast system, each agent is responsible for scanning all incoming communications for whether or not an agent should act or account for the data. A more refined version of the multicast strategy is the publish/subscribe paradigm. In this system, agents can create a chat room, usually called a channel, and publish information to it, in the form of messages. Only those agents that have been subscribed to a particular channel will receive the *messages*. This solution is more flexible, especially when the framework hosts many agents. Other, less frequent strategies include *a direct* communication where data can only be sent to specific agents, or, for some systems, no communication ability exists at all.
- **Resource Management:** Exhausting the local system's processing power and memory resources is a significant risk when running many agents on one system, which, by definition, run continuously and all at the same time. Some frameworks take control of distributing the available system resources (i.e., memory, CPU, disk space, etc) and will either preventively shut down agents using too many resources or simply deny access to them. Unfortunately, the frequent monitoring of the system required to schedule the available resources results in a fairly significant CPU overhead and sometimes impedes the flexibility of the framework. For example, NOMADS uses a modified version of the Java runtime environment to implement its monitoring policy, unfortunately causing NOMADS to be out of date, compared to Sun's current version of the Java runtime environment at the time of writing. While many frameworks choose to forego resource management for these reasons, a framework that supports resource management can create a true sandbox for its agents, a place where the agent cannot harm or impact the host computer in any way, thus allowing the safe execution of agents whose code is not currently trusted. Such a sandbox system can enable the ability to run agents shared by other people, even if you don't particularly trust that their systems are free of viruses, for example. In addition to CPU and memory resource management, a proper sandbox system also needs to restrict and monitor access to data sources, such as access to the network and system storage, such as a hard drive. Some programming languages, including Java, have native support for this kind of security measure. As a result, some frameworks exist that implement this aspect of agent frameworks (SeMoA). By itself, this limited form of resource management will prevent direct damage to the local system by, for example, using the computer's network connection to attack a Web site, but can't stop an agent from disabling the host system. Due to the nature of *C++*, no C++ based frameworks support any kind of resource management.

THE STATE OF THE ART

Table 1 summarizes the properties of the currently available Java-based agent frameworks with respect to the following issues (Pantic et al., 2004):

1. Does the developer provide support for the tool?
2. Is the tool available for free?
3. Are useful examples readily available?
4. Is the related documentation readable?
5. Is synchronous agent-to-agent communication (i.e., wait for reply) supported?
6. Is asynchronous agent-to-agent communication (continuing immediately) supported?
7. What is the communication transmission form?
8. Can the framework control agents' resources (e.g., disk or network capacity used)?
9. Can the framework ask an agent to shut down?
10. Can the framework terminate the execution of a malfunctioning agent?
11. Can the framework store agents' states between executions?
12. Can the framework store objects (e.g., a database) between executions?
13. Does a self-explicatory GUI per agent exist?
14. Does the GUI support an overview of all running agents?

A detailed description of agent frameworks 1-5, 7, 9-18, and 20-24 can be found at AgentLink (2004). A detailed description of CIAgent framework is given by Bigus and Bigus (2001). More information on FIPA-OS is available at Emorphia Research (2004). Pathwalker information is provided by Fujitsu Labs (2000). More information on Tagents can be found at IEEE Distributed Systems (2004). Information on the Fleeble Framework is available from Pantic et al. (2004). The chart shows the emergence of certain trends. For example, termination of malfunctioning agents (i.e.: those that take too many or restricted resources) is offered by only a very small number of frameworks, as shown by columns 8 and 10. Another unfortunate conclusion that can be made from columns 3 and 4 is the lack of proper documentation for most frameworks. The learning curve for such frameworks is needlessly high and seems to be a factor contributing to the large selection of frameworks available.

Sharing a framework so that it is used in as many places as possible has many advantages due to the nature of a framework; namely, to serve as a standard for which agents can be written. Hence, a simple learning curve, supported by plenty of examples and good documentation is even more important than is usual in the IT sector.

Table 1. Overview of the available Java-based agent frameworks

Name	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1. Agent Factory	□	▯	□	□	□	□	PP	□	□	□	□	□	□	□
2. IBM Aglets	□	Free	□	□	□	□	PP	□	□	□	□	□	□	□
3. AMETAS	□	▯	□	□	□	□	M	□	□	□	□	□	□	□
4. Beegent	□	▯	□	□	□	□	PP	□	□	□	□	▯	□	□
5. Cougaar	□	Free	□	□	□	□	M	□	□	□	□	□	□	□
6. CIAgent	□	$	□	□	□	□	PP	□	□	□	□	▯	□	□
7. DECAF	□	▯	□	□	□	□	PP	□	□	□	□	□	□	□
8. FIPA-OS	□	Free	□	□	□	□	PP	□	□	□	□	□	□	□
9. Grasshopper	□	▯	□	□	□	□	M	□	▯	▯	□	□	□	□
10. Hive	□	Free	□	□	□	□	None	□	□	□	□	□	□	□
11. JACK	□	$	□	□	□	□	PP	□	□	□	□	▯	□	□
12. JADE	□	Free	□	□	□	□	M	□	□	□	□	□	□	□
13. JAFMAS / Jive	□	▯	□	□	□	□	M	□	□	□	□	□	□	□
14. Kaariboga	□	Free	□	□	□	□	PP	□	□	□	□	□	□	□
15. LIME	□	Free	□	□	□	□	PS	□	□	□	□	□	□	□
16. MadKit	□	Free	□	□	□	□	M	□	□	□	□	□	□	□
17. NOMADS	□	Free	□	□	□	□	None	□	□	□	□	□	□	□
18. OpenCybele	□	Free	□	□	□	□	PS	□	□	□	□	□	□	□
19. Pathwalker	□	Free	□	□	□	□	PP	□	□	□	□	□	□	□
20. SeMoA	□	Free	□	□	□	□	None	□	□	□	□	□	□	□
21. Tagent	□	Free	□	□	□	□	PP	□	□	□	□	□	□	□
22. Tryllian	□	$	□	□	□	□	PP	□	▯	▯	□	□	□	□
23. Voyager	□	$	□	□	□	□	PS	□	□	□	□	□	□	□
24. ZEUS	□	Free	□	□	□	□	PP	□	□	□	□	□	□	□
25. Fleeble	□	Free	□	□	□	□	PS	□	□	□	□	□	□	□

Legend: □ = "yes", □ = "no", ▯ = unknown, PP = Peer to Peer, M = Multicast, PS = Publish-Subscribe

FUTURE TRENDS: SIMPLICITY

Fulfilling the MAS ideal of creating a truly adaptive, autonomous agent is currently impeded by steep learning curves and lack of flexibility in the available frameworks. Hence, a promising new direction for the agent framework area is the drive for simplicity, which serves the dual purpose of keeping the software flexible while making it relatively simple to write agents for the framework. Newer frameworks such as Fleeble forego specialization to try to attain this ideal. The existence of emergent behavior proves that simplistic agents are still capable of being used to achieve very complex results. Frameworks that give its agents only a limited but flexible set of commands while rigidly enforcing the MAS ideal that one agent cannot directly influence another enables the use of such a framework in a very wide application domain, from a control platform for a swarm of robots to a software engineering paradigm to reduce bugs in complex software by increasing the level of independence between parts of the software, thereby offering easier and more robust testing opportunities. Another area in which simplicity is inherently a desirable property is the field of education. The ability to let agents representing the professor or teacher inspect and query agents written to complete assignments by students represents a significant source of time-saving, enabling adding more hands-on practical work to the curriculum. A framework that is simple to use and understand is a requirement for basing the practical side of CS education on writing agents. More information on using agent frameworks as a teaching tool is available from Pantic (2003).

CONCLUSION

Agent framework technology lies at the heart of the multi-agent systems branch of artificial iIntelligence. While many frameworks are available, most differ substantially in supported programming languages, ability to enable agents to travel (mobility), level of resource management, and the type of communication between agents that the framework supports. Emergent behavior, a research area focusing on trying to create complex systems by letting many simple agents interact, along with a need for flexibility, is driving research toward providing more robust and less complex frameworks.

REFERENCES

AgentLink (2004*). http://www.agentlink.org/resources/agent-software.php*

Bigus, J.P., & Bigus J. (2001). *Constructing intelligent agent using Java.* Hoboken, NJ: Wiley & Sons.

Bourjot, C., Chevrier, V., & Thomas, V. (2003). A new swarm mechanism based on social spiders colonies: From Web weaving to region detection. *Web Intelligence and Agent Systems: An International Journal, 1*(1), 47-64.

Emorphia Research. (2004). *http://www.emorphia.com/research/about.htm*

Fujitsu Labs. (2000). *http://www.labs.fujitsu.com/en/freesoft/paw/*

IEEE Distributed Systems. (2004). *http://dsonline.computer.org/agents/projects.htm*

Pantic, M., Zwitserloot, R., & Grootjans, R.J. (2003). Simple agent framework: An educational tool introducing the basics of AI programming. *Proceedings of the IEEE International Conference on Information Technology: Research and Education (ITRE '03),* .

Pantic, M., Zwitserloot, R., & Grootjans, R.J. (2004). Teaching introductory artificial intelligence using a simple agent framework [accepted for publication]. *IEEE Transactions on Education.*

Russell, S., & Norvig, P. (2003). Artificial intelligence: A modern approach. Upper Saddle River, NJ: Pearson Education.

Tambe, M. (1998). Implementing agent teams in dynamic multiagent environments. *Applied Artificial Intelligence, 12*(2-3), 189-210.

KEY TERMS

Agent Framework: A software agent framework is a program or code library that provides a comprehensive set of capabilities that are used to develop and support software agents.

Autonomous Software Agent: An agent with the ability to anticipate changes in the environment so that the agent will change its behavior to improve the chance that it can continue performing its intended function.

Distributed Computing: The process of using a number of separate but networked computers to solve a single problem.

Emergent Behavior: The behavior that results from the interaction between a multitude of entities, where the observed behavior is not present in any single entity in the multitude comprising the system that shows emergent behavior.

Heterogeneous Agents: Agents of a multi-agent sSystem that differ in the resources available to them in the problem-solving methods and expertise they use, or in everything except in the interaction language they use.

Homogeneous Agents: Agents of a multi-agent system that are designed in an identical way and have a priori of the same capabilities.

Multi-Agent System (MAS): A multi-agent system is a collection of software agents that interact. This interaction can come in any form, including competition. The collection's individual entities and the interaction behavior together comprise the multi-agent system.

Software Agent: A self-contained piece of software that runs on an agent framework with an intended function to accomplish a simple goal.