A New Classification Scheme for Software Agents

Aaron Hector and V. Lakshmi Narasimhan
School of Electrical Engineering and Computer Science,
University of Newcastle, Australia
Email: {ahector, narasimhan}@cs.newcastle.edu.au

Abstract

The field of software agents is a broad and rapidly developing area of research, which encompasses a diverse range of topics and interests. In order to study the various methodologies for agent design, a comprehensive classification scheme is required. This paper identifies the key aspects of software agents, then provides an overview of existing ontologies, and combines the best aspects of these schemes to propose a new all-inclusive classification scheme. In order to illustrate the classifications, the JACK Intelligent Agents architecture is described in the context of the scheme.

1. Introduction

Agent technology is a rapidly expanding area that encompasses many disparate areas of research, and offers several fundamentally different design approaches. The rapid growth of this field in the past decade has occurred in parallel with the evolution of the World Wide Web, as multi-agent systems show great promise for exploiting massively distributed systems such as the Internet. Despite the popularity of multi-agent systems, there is very little consensus about what exactly constitutes a software agent. Research into agent-based systems has been quite diverse [1], which makes it difficult to form a comprehensive definition for a software agent.

While a range of different approaches have been taken for agent design, several key features that are shared among all approaches to agents. The fundamental feature of software agents is *autonomy*. Like their human counterparts, software agents must be able to act on behalf of some other party, be it a person or another agent. To do this effectively, some degree of autonomy is required. Hence, agents must be able to take action when necessary without human interaction.

As a result of their autonomy, software agents must run *continuously*. Unlike much conventional software, which performs a fixed task then terminates, agents run constantly. This allows agents to monitor the current situation and take appropriate action when required. Agents also possess *social ability*, that is, the ability to interact with other agents. The real advantages of software agents come not from individual agents, but from communities of interacting agents.

A number of existing surveys and classifications of software agents have been presented previously. However, these are generally focused on a specific subset of software agents. This paper builds upon these to develop a comprehensive classification that encompasses a broad variety software agents. In addition, it will also serve as an introduction to the essential concepts of software agents.

The remainder of this paper is organized as follows: *Section 2* provides a brief overview of several existing classifications, highlighting the novel or useful attributes of each. S*ection 3* describes a new classification scheme which builds upon these classifications to create an ontology that accommodates the various streams of agent research.

2. Existing Classifications

Several classification schemes exist for software agents, with many of them focusing on either a particular domain or on a specific type of agent. These taxonomies will be explained below, then a new classification scheme, combining the best aspects of these will be presented.

A comprehensive typology of agents has been presented by Nwana in [1], where he identifies an "agent" as a *meta-term*, covering a range of agent types. The three primary attributes that agents *should* exhibit are identified as autonomy, learning and cooperation; however these are not proposed as being necessary. Further, two categories of agents are also defined, static versus mobile and deliberative versus reactive. The first category refers to the ability of agents to move around a network, while the second refers to whether they engage in planning and reasoning based upon an internal symbolic reasoning model.

A taxonomy of agents with special focus on industrial applications, especially manufacturing, has been proposed by Parunak [2], individual agents are classified according to two factors: their function, and their architecture. The division of agents according to their functionality is specific to manufacturing, and hence agents are divided into two categories: production and design. Agent architectures are classified based on two properties: diversity of agents, and sophistication of reasoning. The first property identifies whether agents within a system are based upon the same architecture, or have differing architectures, but having a common communication mechanism. The second property identifies the mechanism that the agent uses for reasoning and/or reacting to its environment. This ranges from pure reaction to pure planning.

Another classification is presented by Franklin and Graesser [3], all agents are identified as being reactive, autonomous, goal-oriented and temporally continuous, with the following attributes being optional: communicative, learning, mobile, flexible (non-scripted), and character (personality). From these properties, a "natural kinds taxonomy of agents" is presented.

Wooldridge and Jennings [4] have produced another classification of software agents. A weak notion of agency is proposed, defining an agent as a hardware or software system with the following properties: autonomy, social ability, reactivity and pro-activeness. One notable aspect of this definition is that it requires an agent to both react to its environment, and exhibit goal-directed behavior, i.e. to be both reactive and pro-active. This is in contrast to several other classifications listed above, which consider purely reactive and purely goal-oriented systems within their classifications.

3. Proposed Classification

In this section, we combine the above classification schemes to form an inclusive ontology, which takes into account the various sub-fields of software agents. The core attributes of software agents are autonomy, temporal continuity and social ability. Agents must be able to run independently, with little or no human intervention, therefore autonomy is a necessary property of agency. Temporal continuity is required, as agents must run continuously, rather than simply perform a task, and terminate.

Agents must also possess some form of social ability. The real advantages of software agents come not from individual pieces of software acting in isolation, but from communities of interacting agents.

In addition to the core attributes, agents may be classified according to the following features:

- Pro-activeness
- Adaptiveness
- Mobility
- Collaboration
- Veracity
- Disposition

Each of these features may be further sub-divided into a list of properties, as explained below. The JACK^{TM} Intelligent Agents architecture will be used to illustrate the classification scheme.

3.1 JACK Intelligent Agents

JACK Intelligent Agents (JACK) [5], developed by Agent Oriented Software, is an *Agent Oriented* development environment that builds upon the Java programming language, providing extensions to implement agent behavior [6]. Agents built with the JACK development environment are compiled to standard Java code before being executed. JACK is based upon the Belief Desire Intention (BDI) model of artificial intelligence, which provides a high degree of autonomy and pro-activeness.

An agent template showing some of the extensions that JACK provides to Java is shown in *Fig. 1*.

```
agent AgentType extends Agent [implements
InterfaceName]
{
  // Knowledge bases used by the agent
  // are declared here.
  #private data BeliefType
      belief_name(arg_list);

  // Events handled, posted and sent
  // by the agent are declared here.
  #handles event EventType;
  #posts event EventType reference;
  #sends event EventType reference;

  // Plans used by the agent are
  // declared here. Order is important
  #uses plan PlanType;

  // Capabilities that the agent has
  // are declared here.
  #has capability CapabilityType
    reference;

  // other Data Member and Method
  // definitions
}
```

Figure 1: Example JACK agent template. From Jack Agent Practicals by Agent Oriented Software [7].

3.2 Pro-activeness

An agent's pro-activeness refers to how it reacts to - and reasons about - its environment, and how it pursues towards its goals. Considering that the purpose of an agent is to autonomously and continuously perform a given set of tasks on behalf of a requester, the approach that the agent takes toward achieving goals is central to its performance. An agent's pro-activeness may be characterized as one of the following:

PURE REACTION: Pure Reaction is the simplest form of behavior, and involves directly reacting to stimuli in the agent's environment, by mapping an input from their sensors directly to an action. This approach has two principal advantages, namely, the agent can react quickly to external events, and it greatly simplifies the process of designing agents. Despite the apparent simplicity of reactive agents, complex behaviors can evolve from interaction with complex environments. Brooks' devised a reactive framework called the subsumption architecture [8] for physical robots, which uses layers of reactive control systems to achieve complex behavior.

PURE PLANNING: Pure planning lies at the other end of the spectrum, and involves agents taking a purely planning, or goal-oriented, approach to achieve their goals. This approach relies upon utilizing planning techniques from traditional AI to identify tasks that need to be performed in order to satisfy the goals of the agent. This approach allows flexibility in the pursuit of goals, but is often slow. The dominant technique for goal-directed agent behavior is the "Belief, Desire, Intention" (BDI) model [9].

HYBRID: Hybrid agents combine these two techniques, incorporating both reactive and planning components. This approach merges the rapid response of reactive agents with the sophisticated reasoning of planning agents, and is therefore widely viewed as the superior approach to agent design. In fact, the notion of agency presented by Wooldridge and Jennings [4] identifies both reactivity and pro-activeness as necessary attributes of all agents. Parunak [2] divides hybrid agents into two classes: "reaction overridden by plan", where the planner may overrule the reactive component if it disagrees with it, and "reaction modified by plan", where the planner can reconfigure the reactive component to behave differently in the future. The latter technique requires a degree of adaptiveness within the agent.

JACK offers a hybrid approach, with two categories of events that may be performed [7]. *Normal Events* correspond to a reactive approach, triggering an immediate response. In contrast, *BDI Events* use a pro-active (goal-directed) approach, including reasoning about plan selection.

Proactiveness is one of the main features of the JACK agent system, with the architecture heavily based upon the BDI model. In order to facilitate this, JACK provides robust tools for plan specification and selection.

3.3 Adaptiveness

Adaptiveness describes an agent's ability to modify its behavior over time. This is a key attribute that is often associated with agents. In fact, the term "agent" is often taken to implicitly mean "intelligent agents", which combine traditional AI techniques to assist in the process of autonomously performing tasks. Adaptiveness is closely related to pro-activeness, with many pure planning or hybrid systems relying on the ability to adapt. Despite this, not all agents are adaptive, and some only adapt in a limited manner. Adaptiveness may fall into several different categories as noted below:

LEARNING: Learning agents have the capacity to modify their behavior over time in order to adapt their functionality to their environment, and to improve their effectiveness. A wide range of techniques have been applied to learning agents, including memory-based learning, reinforcement learning [10], and Bayesian belief networks [11].

SUBSUMPTION: The Subsumption Architecture allows the designer to add additional "layers of competence" to an agent over time. It was first defined by Brooks [8] as an architecture for autonomous robots, but has since been adapted for software agents [12]. The agent designer can hence expand and adapt the agent's functionality over successive iterations of development. This differs from agent learning, as the adaptation is performed by designers explicitly adding functionality to the agent.

NON-ADAPTIVE: Non-adaptive agents are those that do not modify their behavior over time. As noted by Wooldridge [13], although the discipline of "intelligent agents" largely grew out of the field of AI, not all agents need to be capable of learning. The only intelligence that is required by agents is the capability to make independent decisions, i.e. to act autonomously. While learning is often an appropriate technique for agents to employ, its usefulness will depend on the circumstances in which the agent is being used. In mission critical applications, for example, adaptiveness may be a liability, as it could lead to unpredictable behavior by the system.

CONSTRAINT BASED: Constraint-based agents place restrictions on the agent's capacity to adapt, in order to mitigate the problems associated with learning agents in critical systems. This allows many of the benefits of learning agents, while providing safeguards to ensure that the agent still fulfills critical functions.

The JACK development environment provides some support for adaptiveness in refinement of plan selection, however, learning in the traditional AI sense is not a key foundation of the architecture. Developers may employ advanced learning techniques in developing individual agents, but support for such techniques is not provided in the JACK architecture.

3.4 Mobility

Software agents are well-suited to tasks involving large-distributed networks such as the Internet. Consequently, much of the research into agents has revolved around the concept of mobility. An agent may be:

PHYSICALLY MOBILE: Physically mobile agents are capable of transporting their execution between machines on a network. This provides an attractive mechanism for developing software for distributed systems. Mobility is often implemented in a transparent manner, allowing the agent to continue normal execution as it travels around the network. An example of this approach is the Concordia agent [14], developed by Mitsubishi Electric ITA.

LOGICALLY MOBILE: Logically mobile agents are those which physically execute on a single machine, but access other logical locations, via a network connection. These agents may be thought of as visiting these locations in a conceptual sense, although their actual execution is fixed to a single physical machine. Logically mobile agents provide a suitable mechanism for gathering data from the Internet. A typical example of such an agent is a *web spider* [15], which visits and processes a series of web pages by following hypertext links.

STATIC: Static agents are those do not provide a mechanism for mobility at all.

Unlike other agent systems such as IBM's Aglets [16], JACK Intelligent Agents do not provide support for mobility. However, JACK is Java-based, and consequently, agents developed using it can take advantage of Java's cross-platform capabilities and support for serialization to assist in implementing physical mobility. Virtual mobility may also be implemented at the agent level, although no architectural support is provided for it within the JACK system.

3.5 Collaboration

Collaboration among agents underpins the success of an operation or action in a timely manner. For this reason, agents should possess some form of social ability, and this may be divided into two types:

COMMUNICATIVE: Communicative agents are those that are able to coordinate with other agents by sending and receiving messages using some form of agent communication language. This allows a high degree of collaboration, with social activities such as distributed problem solving and negotiation being possible. Several agent communication languages are available, the most prominent being KQML [17].

NON-COMMUNICATIVE: Non-communicative agents are those that do not engage in formal communication. Although direct agent communication is desirable in many situations, it is possible for agents to collaborate without actual communication taking place. Interaction of agents with resources and their environment may lead to collaborative or competitive behavior emerging. For example, the SWARM [18] agent system provides a framework within which communities of agents can interact. The SWARM system is generally used for simulating social systems, and many such simulations demonstrate collaborative behavior without direct communication. A simple example of this is the *HeatBugs* [19] model.

JACK Intelligent Agents provides a message-based communication framework [6], allowing messages to be passed to another named agent within the system. This provides a simple, but extensible, communications mechanism. Several extensions to JACK are available that provide more advanced support for communication and collaboration. For instance, JACK Teams [20] extends JACK to allow agents to be grouped into teams and sub-teams, which act as separate reasoning entities, supporting the full BDI model, with their own beliefs, desires, intentions, and team-level plans. Another extension, called FIPA JACK [21] builds upon JACK's basic communications framework to provide a FIPA compliant communications infrastructure.

3.6 Veracity

Collaborating agents, whether communicative or non-communicative, may attempt to deceive other agents via their messages or behavior. Agents may hence be classified by their veracity:

TRUTHFUL: Truthful agents are those that do not attempt to intentionally deceive other agents. In a *closed environment*, where the veracity all agents is guaranteed, negotiation and interaction is greatly simplified. If an agent indicates that it can provide a service, other agents can assume that it will make an attempt to provide it. If an agent provides information to assist in satisfying a goal, all other agents can be reasonably certain that this information is correct.

UNTRUTHFUL: Untruthful agents are those that may attempt to deceive other agents either through the provision of false information, or by acting in a deceptive manner. In an *open environment*, where various agents from different vendors compete to achieve their own goals, the issue of deception becomes a factor. In such a system, collaboration becomes much more difficult.

JACK is intended primarily for use in closed environments, where all agents are designed to work towards some overall goal, and hence agents designed with it are usually truthful. However, FIPA JACK allows it to be used in open environments such as the Internet, interacting with agents designed using other architectures and methodologies. In such a situation, the designer may need to take into account the possibility of untruthful agents.

3.7 Disposition

The final factor in this classification of agents indicates the "attitude" of the agent toward other agents, and its willingness to cooperate with them. Agents may be classified as:

BENEVOLENT: Benevolent agents are those that always attempt to perform a task when it is requested of them. Wooldridge [13] identifies benevolence as "the assumption that agents share common goals, and that every agent will therefore do what is asked of it". Much like with *truthful* agents, collaboration is greatly simplified in a system that consists entirely of benevolent agents.

SELF-INTERESTED: Self-interested agents are those that act in their own interest, collaborating with other agents only when it is beneficial to do so. Unlike benevolent agents, they cannot be expected to do what is asked of them, so coordinating with such agents becomes a difficult task. In many cases, self-interested agents may be competitive, vying with other agents to achieve a given goal, or acting to secure a better deal than other agents. A typical example of this would be an electronic auction system, such as that mentioned in the example above.

MALEVOLENT: Malevolent agents are those that attempt to inconvenience other agents, or undermine them in some way. Unlike self-interested agents, they do not simply act to achieve their own goals, rather they act in some predefined malicious manner. While the difference between self-interested agents and malevolent ones may simply be a distinction in the goals of the agent, self-interested agents may have a positive impact on the system, while the presence of malevolent agents within a system is unlikely to be of any benefit. Some worms may be seen as a type of malevolent agent.

As mentioned in *Section 3.6*, JACK agents are intended for use in closed environments, and will therefore generally be benevolent. However, when extensions such as FIPA JACK are used to allow interaction in an open environment, the possibility of interaction with self-interested or malevolent agents arises.

4. Conclusion and Future Research

This paper describes a new classification scheme for Agent Technology (summarized in *Figure 2)*. It draws upon several existing ontologies, but provides an all-inclusive classification that takes into account the various aspects of agent research. Due to the wide variety of approaches toward software agents, a simple classification method that assigns each agent to a single grouping is insufficient. A single grouping for mobile agents is inadequate, since almost any combination of the above properties is feasible. In addition, agent architectures may be neutral about certain classifications, for example it may leave the decision of whether an agent based upon the architecture is mobile or not to individual implementations.

The classification presented in this paper allows all manner of agents to be included within the scheme. For instance, unlike some other definitions of agency, e.g. Wooldridge and Jennings [4], our classification does not prescribe that an agent must be both reactive and goal-directed. While the advantages of a hybrid approach is widely noted, this classification allows agents that are purely reactive or purely goal-directed to fit within its definition of agency.

Apart from simply providing a mechanism to analyse and catalogue agents, this classification provides an attractive method for determining how well various agents will interact within a system. Agents that share a large number of features will be better able to collaborate; communicative agents will perform much better in an environment with other agents that share this trait, while static agents will do poorly in an environment designed for mobile agents. An agent that is benevolent and assumes that other agents share the same trait will not do well in an environment with self-interested agents. Determining which types of agents work well with others will be of increasing importance as large-scale multi- agent systems become more widespread across the Internet. This will also become an issue when designing standardized agent architectures. As a consequence, analyzing issues relating to security and certification of agents are very important.

While this paper attempts to define an inclusive classification method for agent-based systems, it is by no means definitive. Further work needs to be done on refining the categories, and reaching consensus on an ontology for agent-based systems. To assist with this, the authors are currently preparing a survey paper which catalogues state-of-the-art agent-based systems into the categories as listed above.

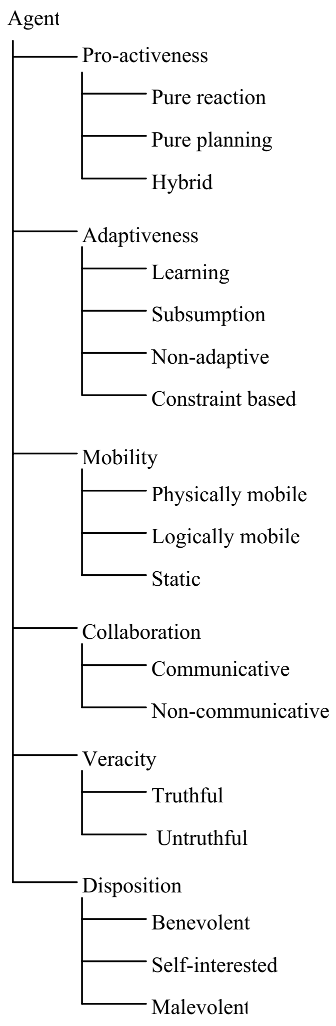

Figure 2: Overview of classification scheme

Bibliography

[1] H. S. Nwana, "Software Agents: An Overview," *Knowledge Engineering Review*, vol. 11, pp. 205-224, October/November 1996.

[2] H. V. D. Paranuk, "Applications of Distributed Artificial Intelligence in Industry," in *Foundations of Distributed Artificial Intelligence*, 1996, pp. 139-164.

[3] S. Franklin and A. Graesser, "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents," in *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*. Springer-Verlag, 1996.

[4] M. J. Wooldridge and N. R. Jennings, "Intelligent Agents: Theory and Practice," *Knowledge Engineering Review*, vol. 10, pp. 115-152, Jun 1995.

[5] Agent Oriented Software Pty. Ltd., "JACK Intelligent Agents."

[6] Agent Oriented Software Pty. Ltd., *JACK Intelligent Agents - Agent Manual*, 4.1 ed, 2004.

[7] Agent Oriented Software Pty. Ltd., *JACK Intelligent Agents - Agent Practicals*, 4.1 ed, 2004.

[8] R. A. Brooks, "A Robust Layered Control System For A Mobile Robot," *IEEE Journal of Robotics and Automation*, vol. RA-2, pp. 14--23, March 1986.

[9] A. S. Rao and M. P. Georgeff, "BDI agents: from theory to practice," in *Proceedings of the First Intl. Conference on Multiagent Systems*. San Francisco, 1995.

[10] R. Kozierok and P. Maes, "A Learning Interface Agent for Scheduling Meetings," in *Proceedings of the 1st International Conference on Intelligent User Interfaces*, 1993, pp. 81-88.

[11] Yi~Shang, H. Shi, and S.S. Chen, "An Intelligent Distributed Environment for Active Learning," presented at ACM Journal of Educational Resources in Computing, 2001.

[12] H. Song, S. Franklin, and A. Negatu, "SUMPY: A Fuzzy Software Agent," presented at ISCA Conference on Intelligent Systems, 1996.

[13] M. Wooldridge, "Agent-Based Software Engineering," *IEE Proceedings Software Engineering*, vol. 144, pp. 26-37, 1997.

[14] Mitsubishi Electric ITA, "Concordia: An Infrastructure for Collaborating Mobile Agents."

[15] C. Cesarano, A. d'Acierno, and A. Picariello", ""An intelligent search agent system for semantic information retrieval on the internet"," in *"Proceedings of the fifth ACM international workshop on Web information and data management"*, 2003, pp. 111-117.

[16] IBM Corp., " The IBM aglets workbench, http://www.trl.ibm.co.jp/aglets/," 1996.

[17] T. Finin, R. Fritzson, D. McKay, and R. McEntire, "KQML as an Agent Communication Language," presented at The Proceedings of the Third International Conference on Information and Knowledge Management, 1994.

[18] M. Daniels, "An open framework for agent-based modeling," 2000.

[19] Swarm Development Group, "SWARM Heatbugs, http://www.swarm.org/examples-heatbugs.html."

[20] Agent Oriented Software Pty. Ltd., *JACK Intelligent Agents - Teams Manual*, 4.1 ed, 2004.

[21] K. Yoshimura, "FIPA JACK: A plugin for JACK Intelligent Agents. Technical report," 2003.