The Open Agent Architecture

Adam Cheyer, VerticalNet
acheyer@verticalnet.com
David Martin, SRI International
martin@ai.sri.com

Overview

SRI's Open Agent Architecture (OAA), a research framework for constructing agent-based systems, makes it possible for software services to be provided through the cooperative efforts of distributed collections of autonomous agents. Communication and cooperation between agents are brokered by one or more facilitators, which are responsible for matching requests, from users and agents, with descriptions of the capabilities of other agents. Thus, it is not generally required that a requester (user or agent) know the identities, locations, or number of other agents involved in satisfying a request. Facilitators are not viewed as centralized controllers, however, but rather as *coordinators*, as they draw upon knowledge and advice from several different, potentially distributed, sources to guide their delegation choices.

OAA is structured so as to minimize the effort involved in creating new agents and "wrapping" legacy applications, written in various languages and operating on various platforms; to encourage the reuse of existing agents; and to allow for dynamism and flexibility in the makeup of agent communities. Distinguishing features of OAA as compared with related work include extreme flexibility in using facilitator-based delegation of complex goals, triggers, and data management requests; agent-based provision of multimodal user interfaces; and built-in support for including the user as a privileged member of the agent community.

Goals

The goals of OAA may be categorized under three main headings, as follows:

Versatile mechanisms of interoperation and coordination

- Provide flexibility in assembling communities of autonomous service providers—both at development time and at runtime.
- Provide flexibility in structuring cooperative interactions among the members of a community of agents.
- Impose the right amount of structure on individual agents.
- Include legacy and "owned-elsewhere" applications.

Human-oriented user interfaces

- Provide conceptually natural means of interacting with multiple distributed components.
- Provide integral support for the construction of multimodal interfaces, including the handling of users' requests expressed in natural language.
- Treat users as privileged members of the agent community.
- Support collaboration (simultaneous work over shared data and processing resources) between users and agents.

Realistic software engineering requirements

- Minimize the effort required to create new agents, and to wrap existing applications.
- Encourage reuse, of both domain-independent and domain-specific components.
- Support lightweight, mobile platforms.
- Minimize platform and language barriers to the creation of new agents, as well as wrapping of existing applications.

System Description

OAA is a domain-independent framework, from which a wide variety of systems can be built. (A sampling of systems built using OAA is given below.) The framework consists of the facilitator agent and libraries, in several languages, which can be used in constructing application agents.

Each OAA-based system includes one or more facilitators, each coordinating the communications and activities of a set of agents. The facilitator is itself a specialized server agent, responsible for coordinating agent communications and cooperative problem-solving. In some systems, the facilitator is also used to provide a global data store for its client agents, which allows them to adopt a blackboard style of interaction. Larger systems can be organized using multiple facilitators, with various communication and cooperation patterns possible between the facilitators.

Agents that are *not* facilitators are referred to as *client* agents because each acts (in some respects) as a client of some facilitator, which provides communication and other essential services for the client. When invoked, a client agent makes a connection to a facilitator, known as its *parent facilitator*, and informs it of the services it provides. Using the brokering services of its parent facilitator, a client agent can provide several types of services, including procedural goal fulfillment, maintenance and querying of data stores, and setting and responding to triggers of several types. The declarations, requests, and maintenance commands associated with all of these types of services are formulated and handled in a consistent, integrated fashion, using the Interagent Communication Language (ICL). ICL is the interface, communication, and task coordination language shared by all agents.

Requests for services provided by other agents are sent to the requester's parent facilitator, which, in turn, delegates them to one or more providers. Requests can be accompanied by a variety of parameters, which serve as advice to the facilitator regarding the desired handling of the request. For example, it can be specified whether, in the presence of multiple agents that can satisfy a request, they should be employed in series or in parallel. Procedural goals and data queries can be simple or compound. Compound requests are composed using Prolog-like operators for conjunction, disjunction, conditional execution, and so forth. With compound requests, parameters can accompany the entire request or any of its subrequests.

OAA triggers provide a general mechanism for requesting that some action be taken when some set of conditions is met. Each agent can install triggers either locally, on itself, or remotely, on its facilitator or peer agents. There are four types of triggers: *communication* triggers "fire" when a message matching a given pattern is sent or received; *data* triggers fire when a data store is updated in some specified manner; *task* triggers fire when some application-specific condition is met; and *time* triggers fire at a fixed time, or at fixed intervals. Triggers are used in a wide variety of ways within OAA systems, for example, for monitoring external sensors in the execution environment, tracking the progress of complex tasks, or coordinating interagent communications that are essential for the synchronization of related tasks.

It is important to note that making a request of the system does *not* require that the requester specify (or even know of) a particular agent or agents to handle the call. While it is possible to specify one or more agents to handle a request, in general it is advantageous to leave these delegation choices to the facilitator.

When a facilitator receives a request, its job is to construct a goal satisfaction plan and oversee its satisfaction in the most appropriate, efficient manner that is consistent with the specified advice. Programming in this style greatly reduces the hard-coded dependencies among components that one often finds in more traditional distributed frameworks.

Client agents may be loosely categorized as follows:

- *Application agents* are usually specialists that provide a collection of services of a particular sort. In some cases, these services will be domain independent and highly reusable (such as speech recognition, natural language processing, email, and some forms of data retrieval and data mining); in others, they will be user specific or domain specific (such as a travel planning and reservations agent). Application agents may be based on legacy applications or libraries, in which case the agent may be little more than a wrapper that calls a pre-existing API.
- *Meta-agents* are those whose role is to assist the facilitator agent in coordinating the activities of other agents. While the facilitator possesses domain-independent coordination strategies, meta-agents can augment these by using domain- and application-specific knowledge or reasoning (rules, learning algorithms, planning, and so forth).
- *User interface agents* play an extremely important and interesting role in many OAA systems. In some systems, these agents are implemented as collections of "micro-agents", each monitoring a different input modality (point-and-click, handwriting, pen gestures, speech), and collaborating to produce the best interpretation of the current inputs.

Systems Built Using OAA

OAA has been used to implement more than thirty applications integrating such diverse technologies as image processing, speech recognition, multiuser collaboration, text extraction, planning, and virtual reality. The table gives a partial list of OAA-based applications, with brief descriptions of their functionality.

Automated Office [Cohen94]	Mobile interfaces (PDA with telephone) to integrated community of commercial office applications (calendar, database, email) and AI technologies (speech recognition, speaker identification, text to speech, natural language interpretation and generation)
Unified Messaging	Adaptable, ubiquitous access to email, fax, voice, and Web messages and services
Multimodal Map [Cheyer95]	Pen/voice interface to distributed Web data
InfoWiz	Animated voice interactive interface to the Web
ATIS-Web	Try out a live demo of speech recognition over the Web! Available at http://www.speech.sri.com/demos/atis.html
CommandTalk 1997. [Martin99] D.L. Martin, A.J. Cheyer and D.B. Moran, "The Open Agent Architecture: A Framework for Building Distributed Software Systems". Applied Artificial Intelligence, Vol. 13, Nos. 1-2, pp.21-128. January-March, 1999.	Spoken-language interface for controlling simulated forces

[Moore96]	
Spoken Dialog Summarization	Real-time system for summarizing human-human spontaneous spoken dialogs (Japanese)
Language Tutoring	Speech recognition for foreign language learning, incorporating user modeling for adaptive lessons
Disaster Response	Collaborative, wireless map-based interface for emergency response teams
MVIEWS [Cheyer98]	Integrating speech, pen, natural language, image processing, and other technologies for the video analyst
InfoBroker [Martin97]	Mediated facilitation of heterogeneous structured and semistructured (Web) datasources
Rental Agent	Monitors the Web and notifies user when housing classifieds meet user specifications
Agent Development Tools [Martin96]	Guides the agent developer through the steps required to create new agents
Multirobot Control [Guzzoni97]	Team of robots works together on assigned tasks (1st place, AAAI Office Navigation Event)
Surgical Telepresence	Force feedback training simulator for endoscopic surgery -- all physical and virtual entities modeled as OAA agents

Lessons Learned and Future Directions

OAA offers many attractive features to developers of distributed systems, and has proven to be extremely useful in building a wide variety of applications. One reason for this success is the ease with which new agents can be constructed. This, in turn, is made possible by several factors: a relatively small set of requirements that must be met by new agents; a unified set of framework concepts, declarations and interfaces that is consistent across all services (goal satisfaction and data queries, messaging, data management, triggers); and the brokering role played by the facilitator, which dramatically eases the coding needed to manage interactions between agents. Another important benefit of this approach is the flexibility with which communities of agents can be assembled, and the flexibility with which services can be added at runtime, and brought into use without requiring changes to other agents.

At a higher level, we attribute OAA's desirable characteristics to a set of concepts that we call *delegated computing* (on which a paper is forthcoming), a central theme of which is the delegation of requests to the appropriate providers, drawing on distributed sources of guidance, by elements of the framework (rather than by the requester). In OAA, the facilitator manages the delegation choices, but its choices are guided by several types of information, provided by agents (and possibly users) in the community.

Another important source of benefits is the use of a high-level, relatively abstract, and yet simple language (ICL), that is used consistently for all agent declarations, requests, and messages. Among other things, we have found that such languages facilitate the use of natural language and multimodal interfaces, because they provide a natural target for the translation of natural language inputs.

Areas for further investigation include scalability; fault tolerance; more thorough monitoring of task completion and adaptation to subtask failure; richer descriptions of agent capabilities; improved development and runtime tools; and improved facilitation strategies and services.

The use of facilitators offers both advantages and weaknesses with respect to scalability and fault tolerance. For example, on the plus side, the grouping of a facilitator with a collection of client agents provides a natural building block from which to construct larger systems. On the minus side, there is the potential for a facilitator to become a communication bottleneck, or a critical point of failure.

We see three general areas to explore in addressing the issues of scalability and fault tolerance. First, a variety of multifacilitator topologies can be exploited in constructing large systems. It would be useful to investigate which of these exhibits the most desirable properties with respect to both scalability and fault tolerance. Second, it is possible to modularize the facilitator's key functionalities. For example, goal planning (delegation and optimization) can readily be separated from goal execution. Given this, one can envision a configuration in which the execution task is distributed to other agents, thus freeing up the facilitator. Third, we would like to incorporate mechanisms for basic transaction management, periodically saving the state of agents (both facilitator and client), and rolling back to the latest saved state in the event of the failure of an agent.

Summary

The Open Agent Architecture provides a framework for the construction of distributed software systems, which facilitates the use of cooperative task completion by flexible, dynamic configurations of autonomous agents. OAA provides a general approach to achieving cooperation between agents, organized around the declaration of capabilities by service-providing agents, the construction of goals by users and service-requesting agents, and the role of facilitators in coordinating the satisfaction of these goals, subject to advice and constraints that may accompany them. Other features of OAA include facilities for creating and maintaining shared repositories of data, and the use of triggers to instantiate commitments within and between agents.

Additional information about OAA, including papers, descriptions of some OAA-based applications, mailing list information and archives, documentation, and source and binary distributions, can be found at http://www.openagent.com.

References

[Cheyer95] Adam Cheyer and Luc Julia, “Multimodal Maps: An Agent-based Approach”. *In Proceedings of the International Conference on Cooperative Multimodal Communication (CMC/95)*. Eindhoven, The Netherlands, May 1995. Also available at http://www.ai.sri.com/oaa + “Bibliography”.

[Cheyer98] Adam Cheyer and Luc Julia, “MVIEWS: Multimodal Tools for the Video Analyst”. *In Proceedings of the 1998 International Conference on Intelligent User Interfaces (IUI98).* San Francisco, California, January 1998.

[Cohen94] P.R. Cohen, A.J. Cheyer, M. Wang and S.C. Baeg, “An Open Agent Architecture”. In *Proceedings of the AAAI Spring Symposium Series on Software Agents*, ed. Oren Etzioni, pp. 1-8. American Association for Artificial Intelligence, Stanford, California, March 1994. Also available at http://www.ai.sri.com/oaa + “Bibliography”.

[Guzzoni97] Didier Guzzoni, Adam Cheyer, Luc Julia and Kurt Konolige, “Many Robots Make Short Work: Report of the SRI International Moble Robot Team”. *AI Magazine*, Vol. 18, No. 1, pp. 55-64. Menlo Park, CA, 1997.

[Martin96] David L. Martin, Adam Cheyer, and Gowang-Lo Lee, “Agent Development Tools for the Open Agent Achitecture”. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM96),* pp. 387-404. The Practical Application Company Ltd., Blackpool, Lancashire, UK, April 1996.

[Martin97] D.L. Martin, H. Oohama, D.B. Moran and A.J. Cheyer, “Information Brokering in an Agent Architecture”. In *Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*. The Practical Application Company Ltd., Blackpool, Lancashire, UK, April 1997.

[Martin99] D.L. Martin, A.J. Cheyer and D.B. Moran, “The Open Agent Architecture: A Framework for Building Distributed Software Systems”. Applied Artificial Intelligence, Vol. 13, Nos. 1-2, pp.21-128. January-March, 1999.

[Moore96] Robert Moore, John Dowding, Harry Bratt, J. Mark Gawron, Yonael Gorfu and Adam Cheyer, “CommandTalk: A Spoken-Language Interface for Battlefield Simulation”. Technical report. Artificial Intelligence Center, SRI International, June 1996.