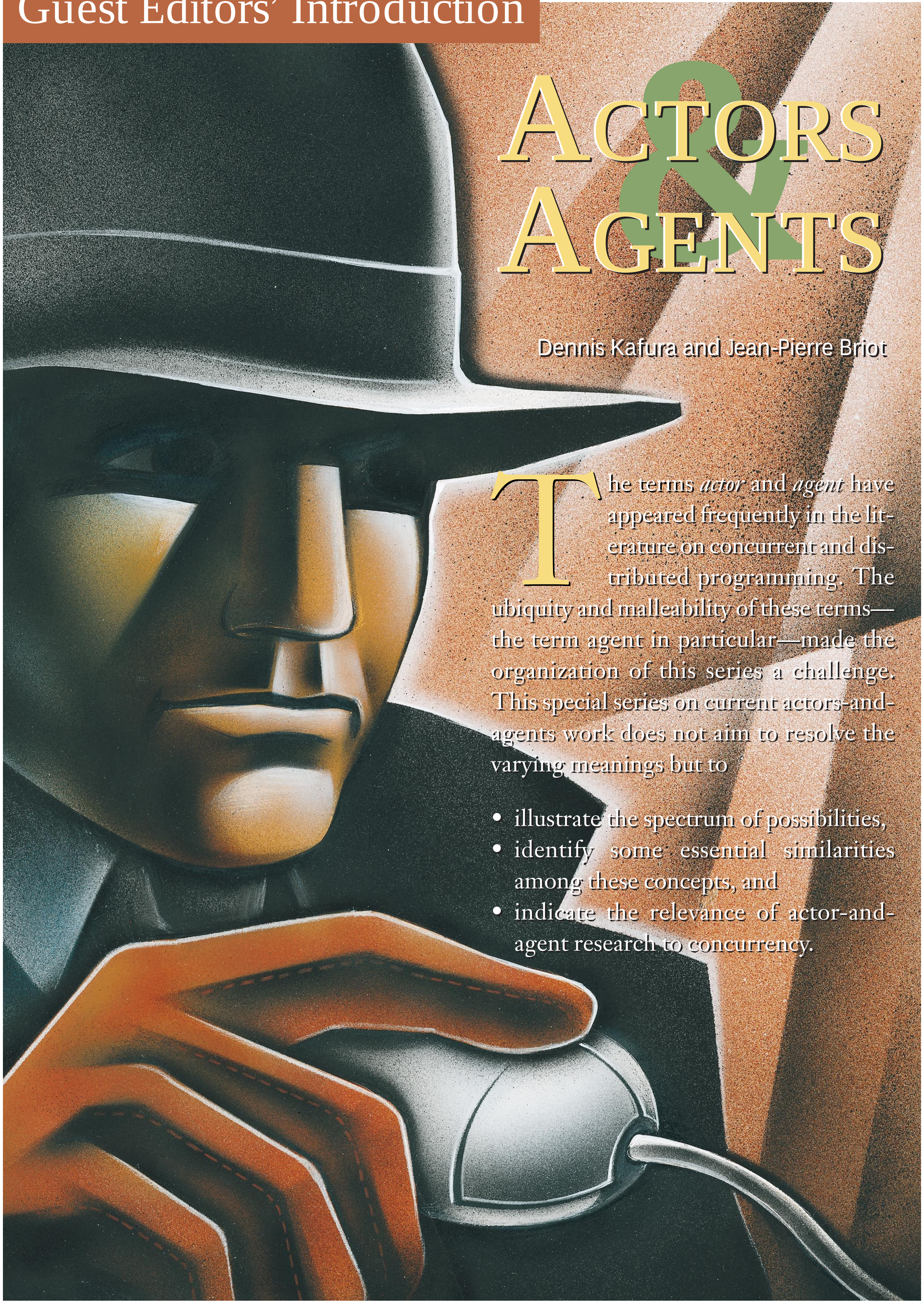
Actors & Agents

Dennis Kafura and Jean-Pierre Briot



The terms *actor* and *agent* have appeared frequently in the literature on concurrent and distributed programming. The ubiquity and malleability of these terms—the term agent in particular—made the organization of this series a challenge. This special series on current actors-and-agents work does not aim to resolve the varying meanings but to

- illustrate the spectrum of possibilities,
- identify some essential similarities among these concepts, and
- indicate the relevance of actor-and-agent research to concurrency.

While distinct, the concepts of actors and agents have both technical and historical connections. In 1977, Carl Hewitt defined an actor as "a computational agent."[1] More recently, Les Gasser and Jean-Pierre Briot conducted an initial study to explore the relations between actors and agents.[2]

What is an actor?

The actor model, defined originally by Hewitt[1] and Gul Agha,[3] is most often described in terms of an actor's structure and operation (see Figure 1). An actor is a message-processing entity that receives incoming messages at a mailbox whose "address" names the actor. An actor's reaction to a message is determined by the actor's behavior when the message is processed. A script defines the behavior. In processing a message, an actor can

- send messages to other actors whose addresses it knows,
- create new actors, and
- determine its own subsequent, or replacement, behavior that will process the next message from the actor's mailbox.

Message passing is asynchronous, nonblocking, reliable, and subject to unbounded delay. Additional constructs or implementation mechanisms can, and often do, introduce ordering and synchrony.

Concurrency arises in the actor model in two ways:

- *inter-actor concurrency*—each actor processes its own messages concurrently with other actors' message processing, or
- *intra-actor concurrency*—in a single actor, the behavior processing one message creates its replacement to concurrently process another message.

The actor model has made essential contributions to the general study of concurrent object-oriented programming.[4] Various object-oriented languages (ranging from Smalltalk[5,6] to C++[7]) have interpreted and implemented the model, and it has played a fundamental role in the design of several experimental languages (for example, ABCL, An object-Based Concurrent Language[8]). Developers have created actor-based applications and systems for concurrent, parallel, and distributed programming on distributed-memory parallel processors, single workstations, and networks of workstations.

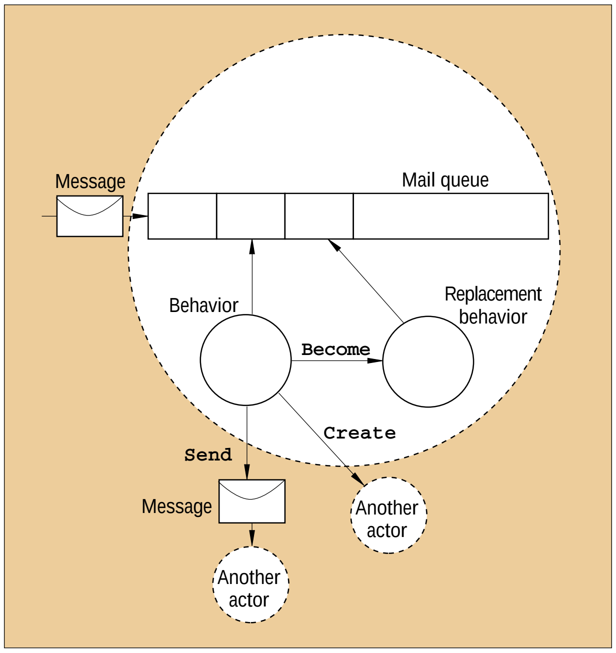


Figure 1. An actor's structure and operation.

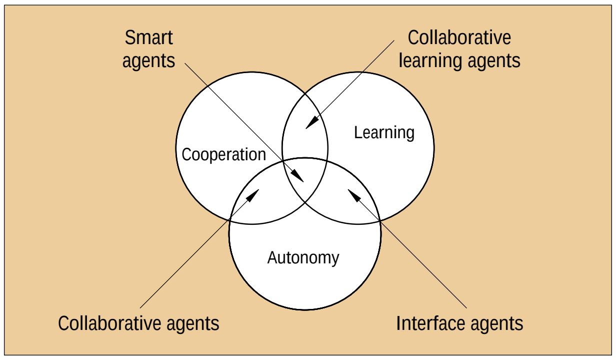


Figure 2. An agent taxonomy.

What is an agent?

The wide use of the term agent defies a simple or universally accepted definition or a comprehensive taxonomy. Attempts to characterize agents[9,10] have contributed to this list of agent attributes:

- autonomy (acts independently),
- continuity (persists over time),
- intelligence (can reason),
- mobility (across machine boundaries),
- personality (possesses a human-like persona),
- adaptability (can learn),
- knowledge (about some domain),
- conversation (is directed at a high level),
- authority (has the rights of its human sponsor), and
- collaboration (interacts with other agents and people).

Figure 2 shows one agent taxonomy,[9] proposed by

Hyacinth Nwana, that involves some of these attributes.

Agent mobility is particularly relevant to concurrency. Java's distributed-programming features have spawned numerous Java-based mobile-agent systems such as Aglets, Concordia, Odyssey, and Voyager.[11] Java's serialization, introspection, platform independence, and remote method invocation have accelerated the development and positioning of these systems as middleware in future systems development.

Agents that appear in a user interface are often termed *assistants*. An assistant is an agent to which a user can delegate a specific task. Such assistants can

- screen electronic mail,
- learn how to perform repetitive tasks,
- arrange meetings,
- gather information, and
- make recommendations based on inferred user preferences or by relating a current user's request to similar requests made by other users.[12]

Trustworthiness and control are two deep issues surrounding the use of agents as assistants. In employing an autonomous assistant, the user must trust the agent to act properly on his or her behalf. For example, to what extent will a user trust a mail-screening agent that can delete messages that it believes to be of no interest to the user? Can an agent be trusted to make the correct inferences? What will cause the user to gain confidence in the agent's abilities? Some argue that agent technology is too ill-defined to warrant complete agent control and that direct manipulation of passive resources is safer and more productive.[13]

Common characteristics

Despite their differences, actors and agents share certain characteristics: identity, autonomy, communication, and coordination. Although these characteristics manifest themselves in different ways, they convey a sense of how agent-oriented or actor-oriented systems differ from systems built on other principles. Systems composed of actors or agents resemble a loose confederation of peer entities more than a rigidly structured configuration of fixed relationships. These systems can be

- flexible, where a system can bring different actor or agent groups into contact without requiring reprogramming;
- adaptable, where a system can dynamically create new actors or agents in response to changing conditions; or
- open, where a standard communication protocol enables interaction among actors and agents that are implemented in different languages or are executing on different systems.

Identity

Each actor or agent is assigned or assigns to itself an *identity*, a means of distinguishing itself from others. An identity allows the actor or agent to communicate and coordinate. The identity of an agent designed to interact with human users is particularly important—the user must have a meaningful way of referring to the agent. With actors, the actor's mail-queue address serves as the identity, or name, of the actor. The actor's identity is determined by the fact that it is the entity that processes messages at a given mail destination. As with actors, an agent's identity can be determined by a name, but a set of the agent's abilities can also specify the identity. Thus, an agent advertises that it can perform a collection of operations or services. Other agents seeking these services would be directed to that agent.

Systems composed of actors or agents resemble a loose confederation of peer entities more than a rigidly structured configuration of fixed relationships.

Autonomy

Autonomy consists of at least three self-centered properties: actors and agents are *self-contained*, *self-regulating*, and *self-directed*. Self-contained implies that the actor or agent has all the machinery needed to fulfill its responsibilities. Although it might interact with others, its capability is essentially complete.

Self-regulating means that the actor or agent can control how and when it reacts to requests. In some cases, it might defer a request—for example, when the state of the actor or agent does not let it safely attempt the requested action. Or, it might completely reject the request if it lacks the required capability or the requester lacks the required authority.

Self-directed implies an ability to react to changing conditions, perhaps simply the passing of time, even without a specific request. In a mechanical sense, this means that an actor or agent has an independent thread of control. In a logical sense, it has a programmed behavior that allows it to know or discover what action to take in the prevailing circumstances.

In practice, actors tend to rely more on preprogrammed, deterministic behavior, while agents factor in reasoning, learning, or planning as part of their self-direction. An actor's autonomy is often described as "reactive," while an agent is often described as "proactive." These terms seem to reflect the views that an actor only responds to a given stimulus (for example, an arriving message), while an agent has an agenda that it actively pursues independent of outside stimuli. In reality, the agent community often uses *reactive* to describe a class of agents that simply reacts to stimuli (thus, close to actors), as opposed to *intentional* or *deliberative* agents, which model and reason about themselves, each other, and their environment. There are also ideas on how to build intentional agents from reactive actors.[14,15]

Communication

Communication enables a large-scale computation to arise from a collection of independent, autonomous entities. Consistent with the autonomy of actors and agents, their communication is most frequently asynchronous and message-based. Asynchrony naturally promotes concurrency, and messaging naturally conforms to the communication structure of distributed-memory parallel processors and to networks of workstations (including the Internet). Standard distributed-systems techniques let actors and agents operate in heterogeneous computing environments. The interaction among actors and agents need not be, and is often not, predetermined. Messages can contain identities—of the sender or of any other parties known to the sender. The receivers of such messages dynamically discover previously unknown partners with which they may thereafter communicate. In this way, systems of actors and agents more intuitively model human organizations and behavior, which establish and use new acquaintances as a natural part of their operations.

An interesting difference between actors and agents is that the recipient of a message sent by an actor is *identity-based* while the recipient of an agent's message can also be *content-based*. Identity-based means that the receiver's identity is specified when the sender emits the message. In this scheme, the message's content is meaningful only to the receiver; no other intermediaries will attempt to parse the message or interpret its content.

Content-based agent communication indicates that the message's structure, and possibly the values in the message, determines the receiver. An intermediary (a *facilitator* in the Knowledge Query and Manipulation Language[16]) can use the structure or content to locate a receiver that can process and respond to the message.

Historically, actor systems have concentrated on communicating data (the parameters to and results from operations), while agents have dealt with communicating through a set of "speech acts" higher-level concepts such as arbitrary descriptions and explicit intentions (for example, request and inform). In the KQML proposal for agent interoperability, in addition to the actual message contents, communication between agents would also specify the intention, the language used for the content description, and the concept hierarchy (ontology).[16]

Coordination

While communication exchanges information (basic data or higher-level descriptions) between actors or agents, coordination lets a set of actors or agents work together by synchronizing their actions, ensuring coherency, and reconciling disparate viewpoints and conflicting intentions.[17] In the actor community, coordination has focused on programming-language issues, and the addressed problems have been closely related to traditional operating-systems synchronization problems (for example, exclusion and resource allocation). A significant issue is *transparency*—inducing the desired coordination among actors without changing, or with minimal changes to, individual actors' programming. A transparent coordination scheme lets the same set of actors interact in different ways, by dynamically changing their coordination milieu. An individual actor can change from one coordination domain to another, or the entire group can adopt a new coordination scheme.

Agent coordination focuses on forming an organization defined by individual agents' roles and protocols.[18,19] An agent

- can rely on the organization to help achieve its own goals and plans,
- might have to help other agents complete their own plans, and
- must respect the organization's interaction rules.

To coordinate agents, the agents might have to articulate their own beliefs and plans and reason about other agents' beliefs or plans. Thus, agent coordination emphasizes knowledge sharing and reasoning, while coordination among actors focuses on synchronization and performance.

This introduction has provided a brief survey of basic actor and agent principles. Articles in the series will highlight more specific details and application domains of actors and agents. In the first article (see page 30), "Tunnel Agents for Enhanced Internet QoS," Hermann de Meer, Antonio Puliafito, Jan-Peter Richter, and Orazio Tomarchio discuss a growing application domain for actor- and agent-based technology: distributed and adaptive network control.

References

1. L. Gasser and J.-P. Briot, "Object-Based Concurrent Programming and Distributed Artificial Intelligence," *Distributed Artificial Intelligence: Theory and Praxis*, N.M. Avouris and L. Gasser, eds., Kluwer Academic Publishers, Dordrecht, The Netherlands, 1992, pp. 81–107.
2. C. Hewitt, "Viewing Control Structures as Patterns of Passing Messages," *Artificial Intelligence*, Vol. 8, No. 3, June 1977, pp. 323–364.
3. G. Agha, "Actors: A Model of Concurrent Computation in Distributed Systems," MIT Press, Cambridge, Mass., 1986.
4. D. Kafura and G. Lavender, "Concurrent Object-Oriented Languages and the Inheritance Anomaly," *Parallel Computers: Theory and Practice*, T.L. Cassavant, ed., IEEE Press, Piscataway, N.J., 1994, pp. 165–198.
5. J.-P. Briot, "Actalk: a Testbed for Classifying and Designing Actor Languages in the Smalltalk-80 Environment," *Proc. European Conf. Object-Oriented Programming* (Ecoop '89), Cambridge University Press, Cambridge, UK, 1989, pp. 109–129.
6. J.-P. Briot, "An Experiment in Classification and Specialization of Synchronization Schemes," *Object Technologies for Advanced Software* (ISOTAS '96), *Lecture Notes in Computer Science*, No. 1049, Springer-Verlag, Berlin, 1996, pp. 227–249.
7. D. Kafura, M. Mukherji, and G. Lavender, "ACT++ 2.0: A Class Library for Concurrent Programming in C++ Using Actors," *J. Object-Oriented Programming*, Vol. 6, No. 6, Oct. 1993, pp. 47–55.
8. A. Yonezawa, *ABCL: An Object-Oriented Concurrent System*, MIT Press, 1990.
9. H.S. Nwana, "Software Agents: An Overview," *Knowledge Engineering Rev.*, Vol. 11, No. 3, Sept. 1996, pp. 205–244; http://www.cs.umbc.edu/agents/introduction/ao/.

10. M.J. Wooldridge and N.R. Jennings, "Agent Theories, Architectures, and Languages: A Survey," *Intelligent Agents: ECAI-94 Workshop on Agent Theories, Architectures and Languages*, Springer-Verlag, 1995, pp. 1–39.

11. J. White, "Mobile Agents," *Software Agents*, J.M. Bradshaw, ed., AAAI Press/MIT Press, Cambridge, Mass., 1997, pp. 437–472.

12. P. Maes, "Agents that Reduce Work and Information Overload," *Comm. ACM*, Vol. 37, No. 7, 1994, pp. 30–40.

13. G. Schneiderman, "Direct Manipulation vs. Agents: Paths to Predictable, Controllable, and Comprehensible Interfaces," *Software Agents*, J.M. Bradshaw, ed., AAAI Press/MIT Press, 1997, pp. 97–106.

14. Y. Shoham, "Agent-Oriented Programming," *Artificial Intelligence*, Vol. 60, No. 1, 1993, pp. 51–92.

15. Z. Guessoum and J.-P. Briot, *From Active Objects to Autonomous Agents*, LIP6 Research Report 1998/015, Laboratoire d'Informatique de Paris 6, Paris 6 Univ.-CNRS, Paris, 1998; ftp://ftp.lip6.fr/lip6/reports/1998/lip6.1998.015.ps.gz.

16. T. Finin, Y. Labrou, and J. Mayfield, "KQML as an Agent Communication Language," *Software Agents*, J.M. Bradshaw, ed., AAAI Press/MIT Press, 1997, pp. 291–316.

17. L. Gasser and R.W. Hill, "Engineering Coordinated Problem Solvers," *Ann. Rev. Computer Science*, Vol. 4, 1990.

18. L. Gasser et al., "Representing and Using Organizational Knowledge in DAI Systems," *Distributed Artificial Intelligence, Volume II*, L. Gasser and M.N. Huhns, eds., Pitman Publishers, London, 1989, pp. vii–xv.

19. A. Drogoul and A. Collinot, "Applying an Agent-Oriented Methodology to the Design of Artificial Organisation: A Case Study in Robotic Soccer," to appear in *J. Autonomous Agents and Multi-Agent Systems*, Vol. 1, No. 1, 1998.

Dennis Kafura is a professor of computer science at the Virginia Polytechnic Institute and State University. His research interests include concurrent, distributed, and parallel systems; object-oriented languages; and Java technologies. He received his MS and PhD in computer science from Purdue University. He is the author of *Object-Oriented Software Design and Construction with C++* (Prentice-Hall, 1998). He is a member of the ACM and IEEE Computer Society. Contact him at the Dept. of Computer Science, Virginia Tech, Blacksburg, VA 24061; kafura@cs.vt.edu; http://www.cs.vt.edu/~kafura/.

Jean-Pierre Briot is a senior researcher at the Centre National de la Recherche Scientifique (CNRS), France. He is also a member of the Laboratoire d'Informatique de Paris 6 (LIP6), where he heads the Objects and Agents for Simulation and Information Systems research team. His research interests include object-based and agent-based models, and architectures and techniques for high-level and adaptive concurrent and distributed computing. He designed Actalk, a framework for various models of actor-based languages and programs, based on Smalltalk. He received his doctorate and his *habilitation à diriger des recherches*, in computer science from the Pierre and Marie Curie University (Paris 6), France. He coedited *Object-Based Parallel and Distributed Computation* (*LNCS*, No. 1107, Springer-Verlag, 1996). He is a member of the ACM. Contact him at LIP6, Paris 6-Case 169, 4 place Jussieu, 75252 Paris Cedex 05, France; jean-pierre.briot@lip6.fr; http://www.lip6.fr/oasis/~briot.