Object-Oriented AI: A Commercial Perspective

Paul Harmon

The coevolution of commercial AI and object-oriented technology are examined here, followed by a brief assessment of future trends and implications.

Computer journals and magazines are currently filled with stories about object technology, just as they were filled with stories about artificial intelligence (AI) and expert systems in the early 1980s. The naive reader might think that expert systems technology had failed and that object-oriented technology was the hot new software approach of the 1990s.

In fact, the story is quite a bit more complex than that. Commercial AI hasn't failed. Indeed, the observant reader of the computer press will notice an increasing number of stories describing expert system applications, neural net applications, natural language applications, etc. Most of the applications described in these articles are either standalone systems or major components in an application. What is even more important is the growing use of AI techniques in applications that no one would normally call an AI application. Microsoft Windows, for example, incorporates "Wizards" and has Inference's CBR engine embedded to provide user help. *Intelligent Software Strategies* newsletter, which has been following the sale of AI products, recently estimated that U.S. knowledge-based system tool vendors sold products and services valued at $159 million dollars in 1994. Thus, although commercial AI is hardly dead, it has not become the huge market some predicted it would in the early 1980s.

If we look beyond more or less pure AI applications, however, it's easy to argue that AI has played a major role in the development of object technology—which is developing into a huge market—and that AI products are rapidly evolving into a key ingredient in many object-oriented (OO) applications.

A Short History of Expert Systems

The first academic expert systems were developed in the late 1970s and the first commercial expert systems were developed in the early 1980s. The first expert systems (ES) were standalone systems designed to function like a human expert—to solve a problem after its symptoms had been described to the system. Expert system-building tools (or shells) were first introduced in 1983 and 1984. ES tools evolved rapidly and a review of the history of tool

vendors provides a good overview of the commercialization of AI in the 1980s [6].

To understand the evolution of expert system-building tools, one should recall what the world of computing was like in the early 1980s. Most computing decisions were in the hands of centralized, corporate database and information systems (IS) groups. These groups relied on mainframes and networks of dumb terminals. They ran applications written in Cobol that were primarily designed to handle back-office, database and transaction processing tasks (e.g., accounting, inventory). Corporate IS groups were under pressure to produce more applications and to cope with maintenance problems associated with the large applications they had previously developed. They wanted help with the tasks they already faced and were reluctant to consider new technology. The IBM PC was introduced in 1981, had very limited memory, and only began to have a significant impact on corporate computing architectures in the mid-1980s.

Most of the innovations that occurred in corporate computing in the 1980s were driven by end users and by departments (e.g., engineering, investment trading, manufacturing, sales) that wanted to solve specific problems and needed new applications developed more quickly than the corporate IS departments could deliver them. Departmental computing groups explored the use of PCs, Fourth Generation Languages (4GLs) and relational databases to create data-entry and report-generation applications. They also tried expert system-building tools. In addition, end users were buying PCs to run spreadsheets and handle word processing.

Many expert system tool companies sold systems to departmental computing groups and proceeded to help the groups to develop interesting systems. As the tool vendors tried to move out of a single department and sell their products to the corporate computing organization, however, they typically ran into serious resistance. Most corporate IS groups regarded AI products with skepticism and, in any case, were not prepared to consider technology that wouldn't run in the mainframe environments.

Changes were also under way in corporate computing. Many corporate computing groups were moving to relational databases. They were also exploring the possibility that structured software development methodologies and CASE tools would increase their productivity and reduce their maintenance problems. These new technologies, however, all fit within the corporate mainframe architecture.

All of the vendors: relational database vendors, CASE vendors, 4GL vendors and expert systems vendors, were trying, each in their own way, to convince IS groups that they should be more flexible. Each, in turn, was faced with the demand that they modify their products to accommodate themselves to the centralized mainframe paradigm.

In successive, overlapping waves, expert system tool vendors modified their products so they: (1) would run on Unix workstations and PCs rather than Lisp machines, (2) were written in conventional languages rather than in Lisp or Prolog, (3) could access data stored in conventional and relational databases, and (4) could run effectively on mainframes. By the end of the 1980s the most successful expert system tool vendors had largely accommodated themselves to the centralized, mainframe paradigm. And then the paradigm collapsed.

It's amazing, in hindsight, how quickly the mainframe paradigm crumbled. In 1990, CASE conferences were dominated by IS people interested in CASE products that would run on mainframes and would promise to support IBM's AD Cycle/Repository. In 1991, attendees at the same conferences were suddenly more interested in PC-based CASE tools that would support client-server development and they were beginning to talk about OO development. Expert system tool vendors whose strength had been in mainframe systems suddenly found themselves shifting gears and trying to offer better PC and Unix versions that would work with Unix database servers. Other vendors who had been trying to develop mainframe versions of their products suddenly de-emphasized their mainframe efforts and focused, instead, on client-server versions of their tools. And all the expert system vendors began to talk about the fact that their tools could support OO development.

A Short History of Object-Oriented Technolgy

Everyone talks about object technology as if they could easily define it. In fact, object technology is very difficult to define because it involves a basic paradigm shift in computing and is involved in almost all aspects of computing. There are OO operating systems, OO middleware products, OO languages, OO methodologies, OO CASE tools, OO 4GL tools and OO databases. People talk about using objects to develop basic definitions of numbers and strings and they talk about encapsulating large Cobol applications as objects. Clearly this technology encompasses a wide variety of different techniques.

Most of the popular accounts of object technology that appeared in the mid- to late-1980s tended to stress the OO language/interface tradition. Object techniques, according to this explanation, were first used in the Simula language and then were further developed by Kay, Goldberg, Robson and others, at Xerox PARC. The first well-known OO language was Smalltalk, which was initially used at Xerox PARC to develop graphical user interfaces [4]. (Early reports tended to talk about Smalltalk as a language, and didn't seem to realize that it was a development environment with extensive class libraries and often functioned as the operating system, as well.)

The Apple Macintosh (developed in Object Pascal by Larry Tesler and others who left Xerox PARC and went to work for Apple) was one offshoot of the Xerox PARC efforts [11]. Various windowing interfaces for workstations were another result. At some point, according to tradition, application developers realized that modularity, in addition to facilitating interface development, made it possible to reuse code, develop applications faster and reduce maintenance problems. Brad Cox probably did the most to promote this point of view with his popular book *Object-Oriented Programming: An Evolutionary Approach* [1]. In the mid-1980s Bell Labs introduced C++ and Objective C and Eiffel soon followed. Later Apple and then Borland made OO versions of Pascal and C++ available. Objective C was used by Next, Inc. to create the NextStep operating system to show what was possible beyond the Macintosh. Microsoft's introduction of Windows 3.0 increased general awareness of graphical operating systems and by 1990 the OO revolution was in full swing.

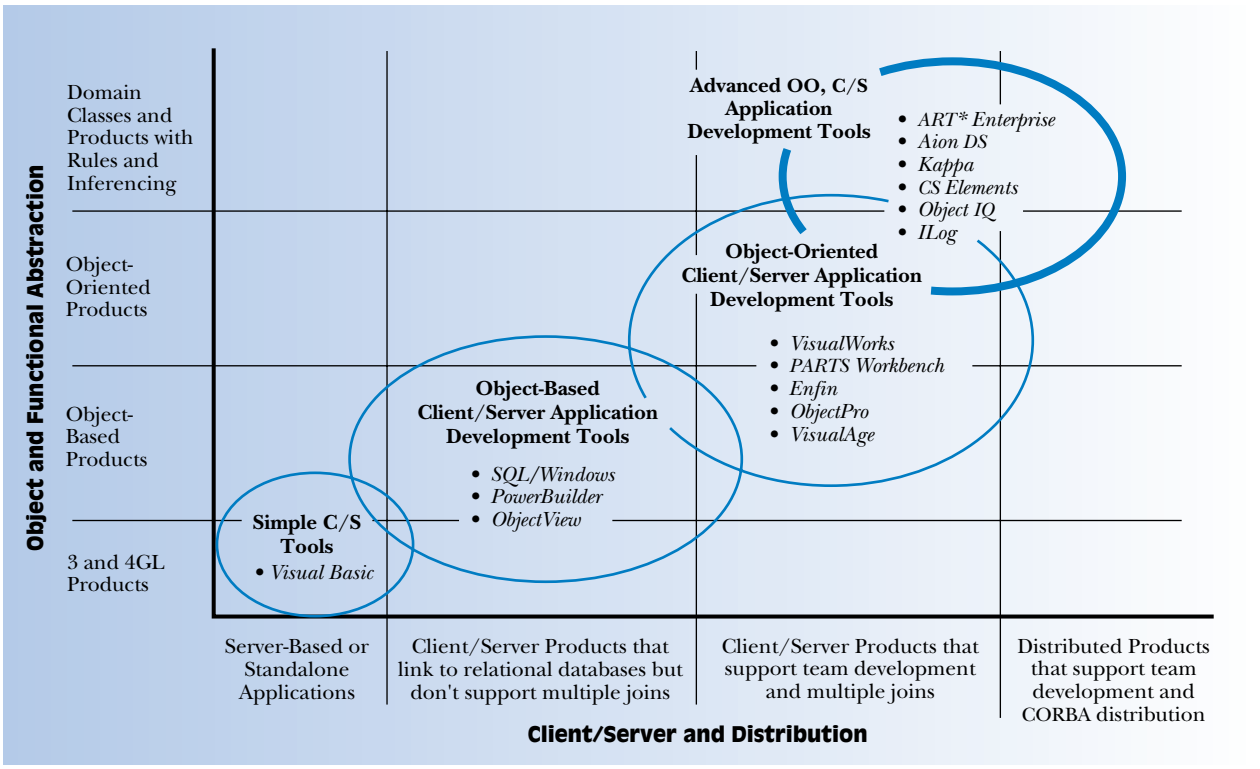
A less well-known account of the development of object technology starts with Marvin Minsky's work on frames and goes through early OO enhancements to Lisp, like Mixins and LOOPS to early expert system tools like KEE [2, 13]. (KEE, Version 1, released in 1984, was a pure OO development tool written in Lisp. It was probably the first commercial OO development tool.) Later versions of KEE added more AI and non-OO features in an effort to make the tool more commercially acceptable for the 1980s [8, 9].

Figure 1.
A matrix comparing OO products based on abstraction and distribution

The earliest accounts of expert systems stressed rules and inferencing, but by the mid-1980s, all of the major expert system-building products had incorporated frames. These early expert system tools used frames in two different ways: (1) to create interfaces and (2) to structure and store knowledge. In addition to their other problems, by the mid-1980s, expert system vendors were trying to explain and sell products that supported inferencing, logic and constraint programming, frame and object-oriented programming and client-server techniques. Only the best programming groups could begin to grasp all of the new technologies, let alone figure out how to integrate them successfully.

Two things that particularly hindered the early expert system tool vendors were (1) a lack of standardization regarding how to represent rules and frames and (2) a lack of a standardized methodology for developing expert systems. The best effort to develop a standardized methodology was undertaken by the European Community (ESPRIT) and resulted in KADS. The first version of KADS, which came out in the mid-1980s, stressed designing systems to solve specific types of decision-making tasks and stressed rules and inferencing. KADS 2, which came out toward the end of the 1980s, was enhanced to incorporate OO techniques [12].

During the same period, Lisp was standardized as Common Lisp and then evolved into CLOS (Common Lisp Object System), which has played a minor, but



Most recent accounts of the OO market *have at least mentioned expert system tools as one way to approach OO development.*

important role in the OO language market. Expert system tools and languages, in turn, led to the development of the first OO databases, (i.e., G-Base) designed to facilitate the storage of knowledge objects. In the 1990s, as OO development became more strongly associated with OO languages, the OO database vendors shifted their emphasis and modified their products to work with C++ or Smalltalk. This is interesting, since neither of those OO languages supports persistent instances, which most expert system tools have relied on since the mid-1980s. Thus, although the AI-derived branch of object technology initially received less attention, recent accounts are beginning to pay more attention to it and a synthesis between the two object traditions is now well under way. In the late 1980s and at the beginning of the 1990s, OO conferences were mostly focused on OO languages and databases. In 1994, most of the shows were focused on OO 4GL client/server tools and OO CASE tools and all the conferences displayed OO products developed by former expert system vendors. Moreover, most recent accounts of the OO market have at least mentioned expert system tools as one way to approach OO development.

To illustrate how AI vendors are selling at the high end of each OO niche, consider Figure 1, an analysis of OO 4GL client/server tools that first appeared in the *Object-Oriented Strategies* newsletter [5]. This analysis emphasizes that tools with inferencing and rules are especially good for logic-intensive, enterprise-wide OO applications. It also emphasizes that few of the current tools support true distributed development or the Object Management Group's Common Object Request Broker Architecture (CORBA). It does not emphasize the difference between objects and frames, which will be considered later.

Developers and companies have been cautious in their adoption of object technology. Most companies began by acquiring object-based products. (Object-based products provide some object components, but lack the full complement of features that most people associate with object-oriented products. Most object-based products, for example, don't support class hierarchies that developers can access or subclass.) In the past two years, as they have learned more about OO technology, many companies have upgraded to true OO products. The advanced OO products will probably remain a small part of the overall OO 4GL niche, however, because most of the products are written in their own internal languages. The OO C/S tools tend to be written in Smalltalk or C++ and make it possible for companies to move classes developed in one tool to other tools or to use them with OO languages. The development of an OO repository might make the advanced OO tools more popular, but this remains to be seen. In the meantime, the emphasis on open systems is definitely limiting the market for most of the advanced OO products.

The OO Language Tradition vs. The OO AI Tradition

Smalltalk and C++ object models have been consistently simpler than the frame-based models. The hierarchies of classes are only used to facilitate the generation of instances and the instances are transient. Multiple inheritance mechanisms are so limited that most OO developers avoid using them. Great stress has been placed on encapsulation.

By contrast, frames were originally conceived as a way to structure knowledge. Thus, frames at any level in a hierarchy may be accessed for information. It's common, for example, for an AI system to begin with a lower-level frame (i.e., an instance), find that it lacks sufficient information, and then move to a higher-level frame and reason on an abstract level. Knowing that a bird is an animal, for example, allows us to assume that it breathes air. The fact that frame systems keep useful knowledge at many levels in their systems has led to more complex inheritance mechanisms and to programming techniques that allow developers to direct messages to frames at various levels in a hierarchy. The original frame systems lacked methods and relied on demons and rules and an inference engine that could operate across all the frames in the systems. Demons were linked to specific target frames. Thus, the use of rules and demons violated encapsulation. (Some early expert system tool vendors tried to encapsulate rules within specific frames and found the process too confusing and inefficient and abandoned it [3].)

In the late 1980s, as companies began to become interested in OO technology, most of the major expert system tool vendors added some kind of methods to their frame systems, making them more like objects in Smalltalk, but rules continued to remain outside the frames. By creating a dual way of handling procedural information, the expert system vendors made their products more complex and developers were faced with still another consideration: when to use methods and message passing and when to rely on rules and inferencing. In addition, the high-end AI-derived products support greater dynamics than languages and tools based on Smalltalk. Thus, most AI tools can dynamically create new classes and new methods when necessary.

In spite of the emphasis that Smalltalk and C++ vendors have put on encapsulation, it has never been considered a defining characteristic of object-orientation. Peter Wegner, in his popular discrimination

Table 1. How AI Techniques Are Being Used In Several OO Market Niches.

OO Market Niche	General Comments	Examples of AI Players
OO Operating Systems	OO operating systems are incorporating intelligent agents and smart checking and indexing components.	Microsoft's Wizards and their use of Inference's CBR technology
OO Languages and Class Libraries	Neither CLOS nor Object Prolog are very popular OOLs, but they provide features that other OOL vendors are incorporating.	Franz, Harliquin, Gold Hill, BIM, Quintus, LPA
OO Networks and Middleware	The CLOS and AI-based development tools support CORBA and have provided some important early examples of enterprise-level distributed systems.	Expersoft's XShell (CORBA)
Components and Frameworks	Several AI vendors offer frameworks for generic KBS system and domain-specific system development.	ILOG's Scheduler Kestrel Institute's work on semantic components
OO Application Development Environments and Tools for Rapid Prototyping	Several KBS tool vendors have repositioned their tools for OO C/S development or created new tools for this niche.	Inference's ART*Enterprise Neuron Data's Smart Elements Trinzic's ObjectPro Hitachi's ObjectIQ
Application Development Environments and Tools for OO CASE	KBS tools have proved good platforms for OO CASE products. Most OO CASE vendors are considering incorporating business rules and inferencing.	IntelliCorp's Object Management Workbench (OMW)
OO Methodologies	OO methodologies have also incorporated business rules and one OO methodology is derived from KBS development.	Martin/Odell's OOIE methodology ILOG's KDSs methodology
OO Databases	Early OO databases were developed to support KBS development and some still include support for rules and will soon incorporate smart search techniques.	Itasca's Itasca ODB Deductive ODBs
OO Tools for special types of applications	In addition to expert system development, OO tools with large AI components include Real Time Simulation Tools, CAD tools, and OO BPR tools.	AutoCAD, Gensym's G2 and ReThink, Carnegie Group's TestBench
Business Objects and OO Applications	Some of the best examples of enterprise-wide OO applications that incorporate business object models of the corporation were developed by KBS vendors.	Boeing's Product Knowledge Manager (Built in KEE. An Object Management Group award winner.) Most large applications that have won IAAI awards.

between OO languages and object-based languages omits encapsulation and includes CLOS among the OO languages [14]. In effect, frames are considered a type of object, and methods can either be encapsulated, or in the environment, as they are in CLOS. At this point, most OO theorists ignore frames and simply stress that AI-derived tools, like CLOS, are simply more complex than most other OO products.

Most corporate OO developers started out in C++ or Smalltalk and are a bit overwhelmed by the complexities of the more sophisticated AI-derived OO tools. As more complex OO applications are developed, however, the more sophisticated OO developers are increasingly looking to AI techniques to help them model problems that are more complex than those commonly discussed in introductory C++ courses. AI developers introduced rules and inferencing, after all, to modularize complex chains of procedural logic. Many OO developers begin by assuming that they are modularizing procedural code when they break a conventional application into methods and associate different methods with different objects. They are modularizing it, of course, but as they move from applications that primarily involve manipulating data to those involving large amounts of knowledge and complex decisions, they find that even methods can get complex. CLOS offers developers the option of using methods with associated pre-conditions and post-conditions to handle more complex situations. At some point, however, it becomes easier to conceptualize the logic as rules and let an inference engine handle the search. In fact, it is only when a developer uses both objects and produc-

tion rules that he or she has really modularized an application in the most systematic manner.

Expert System Vendor Responses to Interest in OO

In the early 1990s, as interest in new expert system-building tools declined and companies became enthusiastic about objects, expert system tool vendors began to reposition themselves. There have been at least four responses:

- Expert system tools as flexible interface development tools (e.g., Neuron Data's Smart Elements)
- Expert system tools as OO 4GL development tools (e.g., Inference's ART*Enterprise)
- Expert system tools as a foundation for OO CASE tools (e.g., IntelliCorp's Kappa)
- Expert system tools as domain specific OO development environments (e.g., Gensym's G2 and ReThink)

It's a bit premature to speculate on how all of these strategies will work out. To date, the domain-specific strategy has been the most successful. Most of the vendors who have chosen to position themselves as purveyors of OO 4GL tools have encountered difficulty. First, they have relied on proprietary Prolog or Lisp-like internal scripting languages. These languages give the tools their power and flexibility, but are not standard OO languages. Second, the tools still contain AI features that make them more complex than the other OO 4GL vendors. These two things combine to make it hard to reuse classes developed in an AI-derived product in other OO contexts. In spite of problems, however, expert system companies have made some progress toward getting themselves recognized as providers of high-end OO development tools.

One of the more interesting developments in OO and CASE is the number of OO CASE vendors who are incorporating some kind of inferencing and rules to allow developers to incorporate "business rules" in applications. The best example of this trend is Object Management Workbench (OMW), an OO CASE product with business rules and an interpreted development environment that is sold by IntelliCorp and James Martin Associates. OMW sits on top of IntelliCorp's ProKappa product [10]. In effect, the power of the AI environment is hidden from the user who develops object models using standard object-diagramming conventions. Production rules are called "business rules" and specified independent of the object model. The power of the AI environment is apparent to users because they can execute diagrams at any time and edit diagrams while they are being executed. To the developer, however, this is simply a nice feature and does not raise the implementation issues that occur when one attempts to field an application developed using some of the advanced OO C/S development tools.

One way to assess the success of ES vendors' efforts to position themselves as OO vendors is to consider how products with both OO and AI characteristics have performed in the contests sponsored by the Object Management Group (OMG) and *Computerworld* at three past ObjectWorld shows in San Francisco. Each year there have been five winners, and each year one of the five winners was developed using an expert system-building tool (i.e., KEE and KappaPC) [14].

As companies move from smaller OO applications to more sophisticated systems, we expect that developers who have used ES tools in the 1980s will encourage their companies to use OO AI tools for tasks involving complex logic.

Other AI Technologies Being Used in OO Products

Those who follow the commercial neural network market, the natural language market, the robotics market or the imaging market are aware that OO techniques have been used in these AI markets and that they are playing roles in various specialized OO markets similar to the role the expert systems vendors are also playing.

Table 1 lists some of niches in the current OO market and comments on some of the ways AI is being used in each niche. In most cases, AI is being incorporated into OO products to make them more flexible. Since the addition of AI techniques tends to make a product more complex, most of the OO/AI products are positioned at the high end of the niche and are being sold to more sophisticated departmental developers. ARPA has awarded a grant to encourage distributed computing. The winning combination includes Expersoft's XShell, an AI tool that also includes a powerful implementation of CORBA. ARPA has also given grants to about a dozen vendors with AI experience to encourage the development of semantic component models. One of the OO databases (Itasca's Itasca ODB) is a Lisp-based OODB derived from work done at MCC.) Some of the OO database people are experimenting with Deductive OO database models.

The Future

Dramatic changes are occurring in the organization of corporations. Companies are trying to position themselves to compete in a worldwide economy. To do this, they are trying to reengineer their internal processes to take advantage of what computers can offer. At the moment, most companies are focused on creating new infrastructures that will facilitate worldwide information exchange and significant gains in productivity.

Object technologies offer a framework on which to rebuild corporate computing. Object technologies will create new operating systems, new networks, new ways of building applications and new ways of storing data. An immense advantage of OO technology is that it can encapsulate old applications, making the transition less painful. More importantly, the move to object technology is just beginning. The complete CORBA standard has just been finalized and won't be implemented until some time this year. This should begin to nudge companies beyond their early, narrowly conceived client/server systems to distributed systems that

will depend on objects and messaging. Products like IBM's SOM, which facilitates the distribution of class libraries and component-standards like Microsoft's OLE2 and CILabs' OpenDoc are becoming available and should, in the course of the next two years, radically change the ideas that most corporate developer's have about reusability. Corporate users "discovered" Smalltalk in 1994 and seem much more excited by Smalltalk than they were by C++. Recently, OO Cobol has been released. In addition, the OO database vendors have agreed on a standard and an OO version of SQL. Approximately 10% of the corporate IS groups were committed to OO development by the end of 1994. We estimate that by the end of 1996 some 40% of the IS groups in the U.S. will be committed to OO technology. In other words, companies are only beginning to get serious about using object technology.

At some point in the future, companies will begin realizing another advantage of object technology; it will provide the matrix into which knowledge and intelligent decision-making techniques can be incorporated. Some developers are already aware that AI can be easily blended with object technology. Most people, however, are too focused on trying to transition from mainframes, Cobol, and structured methodologies to objects and distributed computing to worry about the more advanced capabilities to be obtained from AI-derived OO techniques.

Most of the expert system companies that began in the mid-1980s have tried to reposition themselves as object-oriented, client-server tool vendors. Some have made minor changes to their products and some have created entirely new products to sell to those interested in OO development. This strategy will probably keep some of the expert system companies alive. Taking a long-term view, however, the expert system companies—like the relational database companies, and, indeed, IBM and Microsoft—are caught between their installed base of customers and a new market that has only begun to evolve. Long before most companies get serious about acquiring advanced OO tools, corporate users are going to settle on new languages, new methodologies and work out how to store and use class libraries. Perhaps a vendor like Taligent will set a standard that will then define many subsequent aspects of the ongoing shift to object technologies. Perhaps the object database vendors will set important standards. Once these evolving standards are settled, companies selling products with advanced features will need to work with the new object standards in order to get the attention of corporate developers. The current AI vendors can only influence the direction in which corporations are evolving to a very limited degree, although AI-trained individuals are playing key roles in helping to develop the object standards that we will have to live with in the future.

The expert system vendors of the 1980s helped to start the OO revolution that is slowly sweeping corporate computing in the 1990s. Once the new object technology standards are in place, probably toward the end of the 1990s, it is easy to imagine that corporations will become very interested in techniques that can extend their object infrastructure as they create new applications for their evolving worldwide systems. They will want knowledge and intelligence that can be incorporated into the class libraries they will use for application development. They will want intelligent agents that can search through worldwide networks of databases to locate information. Once they have the infrastructure in place, they will turn to AI-based object techniques to enhance their systems. They will, however, insist on tools that support standard OO languages and methodologies. Those OO AI vendors who have redone their tools to be compatible with the OO standards that emerge during the 1990s will be well positioned to prosper in the late 1990s and beyond. ◻

References

1. Cox, B. *Object Oriented Programming: An Evolutionary Approach.* Addison-Wesley, Reading, Mass., 1986.
2. Fikes, R. and Kehler, T. The role of frame-based representation in reasoning. *Commun. ACM 28*, 9 (Sept. 1985), 904–920.
3. Finin, T. Understanding frame languages. *AI Expert* (Nov. 1986), 44–50.
4. Goldberg, A and Robson, D. *Smalltalk-80: The Language and Its Implementation.* Addison-Wesley, Reading, Mass., 1983.
5. Harmon, P. Object-oriented application development tools. *Object-Oriented Strategies Newsletter 8* (1994), 1–10.
6. Harmon, P. and King, D. *Expert Systems: Artificial Intelligence in Business.* Wiley, NY, 1985.
7. Harmon, P. and Taylor, D. *Objects in Action.* OMG/Addison-Wesley, Reading, Mass., 1993.
8. IntelliCorp. Manuals for versions 1, 2 and 3 of their KEE product.
9. Kehler, T.P. and Clemson, G.D. KEE: The knowledge engineering environment for industry. *Systems and Software 3,* 1 (Jan. 1984), 212–224.
10. Martin, J. and Odell, J. *Object-Oriented Analysis and Design.* Prentice Hall, Englewood Cliffs, NJ, 1992.
11. Schmucker, K.J. *Object-Oriented Programming for the Macintosh.* Hayden Books, Indianapolis, Ind., 1986.
12. Schreiber, G., et. al. Various KADS reports issued by the University of Amsterdam, beginning in 1986. (ESPRIT Project P1098)
13. Stefik, M. and Bobrow, D.G. Object-oriented programming: Themes and variations. *AI Magazine* (1986), 40–62.
14. Wegner, P. Workshop on OO Programming at ECOOP, 1987. Reported in *SIGPLAN Notices 23*, 1 (Jan. 1988).

About the Author:
PAUL HARMON is the founding editor of *Intelligent Software Strategies* newsletter and currently edits *Object-Oriented Strategies* and *BPR Strategies* newsletters. Current research interests include development of markets for new software technologies, and the use of software technologies to transform business organizations. **Author's Present Address:** Harmon Associates, P.O. Box 1198, Inverness, CA 94937; email: Harmon@Mobius.net