An evolutionary approach to application development with object technology

by R. A. Henders

Although object-oriented programming (OOP) is not new, it has only recently begun to gain acceptance among independent software vendors. Reasons for this acceptance vary, from a need for basic data encapsulation, to the promise of code reuse, through problem abstraction as a way of dealing with complexity. Despite the advantages inherent in OOP, obstacles to integration with or replacement of existing systems are significant. This is especially true for Application System/400™ (AS/400™) application vendors, because of a tradition begun with the System/38™ of customers demanding source code. Each independent software vendor (ISV) must determine how to make the transition from procedural to OOP languages in a cost-effective way. This must be done for both ISVs and customers, who have often invested heavily in enhancing and modifying source code to meet their business needs. Acacia Technologies has studied the problem and has adopted a strategy aimed at easing the transition by using a phased approach, starting with encapsulation of AS/400 RPG/400™ functions, continued through relocation of modules where appropriate into a multitiered client/server architecture, with a final target of object-oriented modules communicating in a networked environment. This paper will discuss our approach and the part the San Francisco™ project is expected to play in its implementation.

Although object-oriented programming (OOP) was originally developed in the 1960s,[1] its current level of acceptance in the business community is appropriately characterized by Bruce F. Webster:[2]

> Object technology, as applied to production-grade projects, is still a largely young and somewhat narrow field, especially compared with the well-hammered and heavily used structured development methodologies of the past 25 years—and there are still controversies and arguments over those.

Reasons given for this slow acceptance include a lack of appropriate and adequate tools,[3] concerns about "the value of reuse,"[4–6] and resistance to paradigm shift as described by David Taylor.[7] According to Taylor, a paradigm is

> an acquired way of thinking about something that shapes thought and action in ways that are both conscious and unconscious. Paradigms are essential because they provide a culturally shared model for how to think and act, but they can present major obstacles to adopting newer, better approaches.

Also from Taylor, a paradigm shift is

> a transition from one paradigm to another. Paradigm shifts typically meet with considerable resistance followed by gradual acceptance as the superiority of the new paradigm becomes apparent. Object-oriented technology is regarded by many

advocates as a paradigm shift in software development.

While each of these considerations is valid, our experience has been that a skilled and motivated group of developers will work through tool shortages. This was shown by their willingness to revert to DOS when working with Java** in the Java Development Kit (JDK), supplied by Sun Microsystems. These developers consistently produce good code on a timely basis with or without object-oriented inheritance and the code reuse it offers. Likewise, the potential for career enhancement coupled with interest in new and potentially better technologies seems to overcome resistance to paradigm shift. Our experience has been that the main obstacles in the path to OOP acceptance are (1) the challenge of integrating the new environment with legacy code, at least until a total changeover can occur, and (2) the lack of a foundation of business objects upon which to build. Our approach to dealing with these problems at Acacia Technologies is the subject of this paper.

Our products and development environment

Like many independent software vendors (ISVs) working in the Application System/400* (AS/400*) market, Acacia Technologies has grown from a base of System/38* programs in the early 1980s to millions of lines of code in thousands of programs on the AS/400 today. Internal development and acquisition have resulted in three main manufacturing-oriented products under the Acacia Technologies umbrella. The first is a system aimed primarily at engineer-to-order[8] manufacturers. The second is aimed at mixed-mode[9] manufacturing and operations with large distribution networks. The third is a system that controls all aspects of a warehouse as a stand-alone system or integrated with the other manufacturing products. Acacia Technologies products are installed in more than 2000 systems in 22 countries.

The market has demanded that we and most of our competitors make available source code, which has been modified, for a variety of reasons, by many of our customers. This has typically been due to unique customer requirements that needed to be satisfied more quickly than we or our partners could react. Over time, code-marking standards and source-comparison and -merging utilities have been developed to synchronize our customers' and our own changes in a new software release. The majority of our code remains RPG/400*, ILE (Integrated Language Environment) RPG/400, and AS/400 CLP (Control Language Programming). We have, however, taken advantage of client/server opportunities as well as system APIs, user spaces, and the User Interface Manager[10] where appropriate. In addition, we have integrated personal computer (PC)-based tools and programs utilizing client/server technology and have implemented a graphical user interface option for our customers.

Smalltalk effort. Three years ago we began exploring alternative technologies for dealing with the complexities of today's manufacturing control and planning activities. An important strategic decision for our company, which has for a number of years had an emphasis on manufacturing control and planning software, was to develop an advanced planning and scheduling tool to enhance the product line. Because of the complex nature of this type of system, as well as the dependence on machine-cycle-intensive processes, we decided to develop it using OOP tools and position it on a PC platform.

Smalltalk, from ParcPlace-Digitalk (now ObjectShare), was selected as the language for development. Data were to be downloaded from an AS/400 into an object-oriented database. Objects were to be manipulated on the PC, then updated data sent back to the AS/400. Significant effort was expended in mapping objects from a relational database to an object database and, via transactions and database triggers, back to a relational database. The results of these efforts were very positive, giving us confidence that the same work could be done in other environments. The product was announced as being generally available in August 1996.

Conversion effort. While the development effort was in full stride for the scheduling tool, we were also examining applications on the AS/400. We hoped to move to an architecture that would take advantage of the emerging ILE RPG/400 programming model. Emphasis was given to isolating common business processes from a number of programs and placing them in reusable modules, which were aimed at reducing complexity and strengthening the integrity of the database. Business rules and referential integrity tests were encapsulated in these modules, which acted as servers and were accessible from other AS/400 and PC programs. Currently available servers include:[11]

- Customer Order Server. This transaction server contains all the validation and manipulation needed to add, change, or delete customer orders.

- Product Structure Server. This master file transaction server processes global bill of material requests.
- Purchase Order Server. This extract server consolidates supply and demand information for display.
- Trading Code Server. This master file server contains all the validation and data manipulation that is performed against the trading code file.
- Vendor Master Server. This master file server contains all the validation and data manipulation performed against the vendor master file.

As time progressed, efficiencies in reuse, encapsulation, and performance (when using ILE/RPG static binding) became apparent. Additional functions were added as new servers outside of the database. A generalized date server was added to assist in fulfilling Year 2000 compliance and a currency conversion and display server were planned. Wherever possible, protocols for parameter passing were modeled after the messaging model described by the Open Applications Group (OAG).[12] Acacia Technologies maintains board-level membership in OAG and is committed to adhering to its openness standard.

The two development efforts also brought numerous benefits, reducing development time and complexity, and made us start exploring ways to apply them more broadly across the product line. The ILE/RPG initiative, though important, was limited to common server routines invoked from programs following a traditional program model such as RPG or a client-based development language. The greatest advantage was to be gained in expanding to a fully object-oriented environment.

While the internally developed tools and utilities mentioned earlier have been valuable in our normal operations and may have some ultimate use in a transition to OOP (legacy programs may need to be changed to communicate with objects), we felt that they would not help in implementing a mapping from modules to objects. It was expected that the majority of work involved in moving to OOP would be manual (with some use of automated design and documentation tools). We also felt that, before embarking on such an ambitious project, we should reexamine our choice of OOP language and environment.

Although we saw the advantages of OOP, our experience with Smalltalk made us concerned that there could be scalability issues:

1. Smalltalk evolved for ease of use, hiding complexity in a way that makes it difficult to enhance performance. The design and workings of the Smalltalk virtual machine were not readily available nor was information on platform-dependent interfaces.
2. Smalltalk's dependence on a base image for an application made upgrades tied to use of an object-oriented database difficult, even with the use of Smalltalk link libraries. In addition, versioning tools were not available for the database we had selected.
3. The Smalltalk programming environment is unique, requiring significant effort for developers undergoing retraining and a premium for new hires.

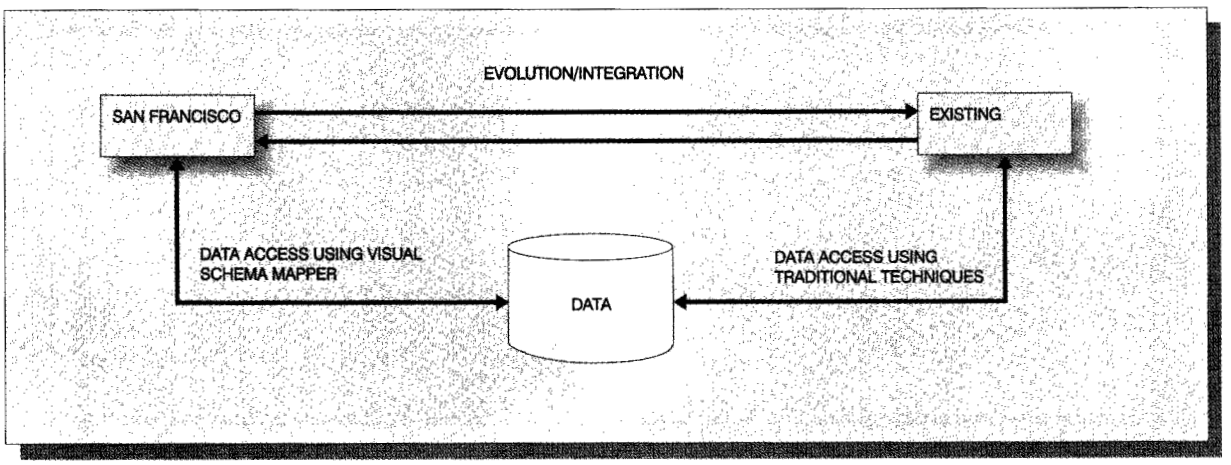
We subsequently investigated and experimented with C++ and felt that it offered performance gains and a more traditional development and delivery environment, but carried with it the potential for increased complexity and a longer learning curve and development cycle for an expanded pool of developers. This held the threat of increasing the time to adoption of a technology (OOP) that offered significant advantages.

Moving to San Francisco

We began examining Java (independent of the San Francisco* project, which is built on a Java base), and saw its potential as a marriage of the best parts of the two generally available object-oriented languages (Smalltalk and C++):

1. The Java language is relatively open, which allows the underlying function to be exploited. This, in turn, gives developers more potential for control and flexibility.
2. Although performance has been and continues to be an issue with Java, indications are that problems in this area will be solved.
3. Development in Java, as carried out in currently available integrated development environments,[13] is more like traditional PC development environments than Smalltalk development.
4. JDBC** support was offered with JDK 1.1, and several object-oriented database vendors have indicated current or future support for Java.
5. The exploding popularity of the language holds the promise of availability of developers with Java skills.

Figure 1 Integrating existing programs with San Francisco



As we learned about the reference group[14] of the San Francisco project, we felt that their commitment to Java for base objects and frameworks was key to our involvement. As described elsewhere in this publication, the San Francisco project has been a combined effort of IBM and ISVs to provide an object-oriented infrastructure with a framework for basic application logic. A design goal has been the delivery of approximately 40 percent of the code for a completed application, with an opportunity for ISVs to provide product-differentiating features in the remaining code. Since we have always had a commitment to the AS/400 as our server platform (and continue to state this as our direction), we were not as interested in the portability of Java or the San Francisco project as we were in the object-oriented language[15] and basic objects and frameworks.

The prospect that base business objects and frameworks would be supplied was particularly important because it supported the basic objectives of the OAG. Without this, domain developers would spend time developing objects that would be essentially the same as the objects built independently by other developers. Spending effort creating common objects, such as business partner, bank, chart of accounts, etc., would be like starting with a graphical point object, expecting that the finished application would have a sophisticated common graphical user interface. It could be done, but the effort, even with code sharing, would be significant and the result would probably be different for each team that started with the same point object.

Of equal weight in our decision was the San Francisco objective for coexistence with existing (legacy) applications. Writing a basically stand-alone application, with ties to the AS/400 database via application programming interfaces wrapped in objects (as we did with the advanced planning and scheduling tool), was recognized as significantly less complicated than moving a full enterprise application to OOP. A complete rewrite to OOP on an enterprise scale, while holding the promise of better response to market demands in the future, adds little short-term value. This significant consumption of resources with little or no immediate return is difficult, if not impossible, to justify. The only economically viable approach is the coexistence of parts of the old and new systems in a controlled migration. Our concerns were answered at several of the reference group meetings.

Figure 1[16] shows the San Francisco project's commitment to integration of existing programs into the general model. Note that Java (San Francisco) and non-Java (existing) applications can coexist, and that stored data can be either persistent objects or in a relational file system. This coexistence is key for transition to OOP in a managed and reasonable fashion over some period of time.

Conversion methodology. Our basic methodology for converting from traditional programs to OOP is described by the following stages:

1. Traditional programs are examined (manually, by automation, or a combination) for sequences of

operations that should be considered for isolation and reuse.
2. After a minimum number of current and anticipated uses are projected for a particular sequence of operations, a new module is designed and changes indicated for existing programs.
3. New modules are created, interfaces documented, and existing programs changed in a coordinated test environment aimed at future release.
4. Traditional programs are again examined and unique business functions (that cannot be placed in reusable modules) are highlighted to ensure that they are carried forward.
5. Elements of both reusable modules and traditional programs are included in a model to ensure consistent implementation and discover additional reuse opportunities.
6. The model is implemented in a phased approach as a combination of a decreasing number of traditional programs and an increasing number of objects.
7. As the transition is being completed, newer emerging technologies are being examined for future integration.

At the present time we are not actively investigating the location of modules in a multitiered environment. Our experience with the request broker provided with San Francisco is that this will be more a tuning than a design issue.

We are actively involved in three separate stages of the methodology. Stage 3 is nearing completion as existing server opportunities are implemented, while Stages 4 and 5 are operating in parallel. A high-level example of our methodology for converting from a server to an object is the replacement of our vendor master server by an appropriate San Francisco object.

Conversion example. Our vendor master server deals with inquiries against and changes to our vendor master file. Valid functions exposed in the server are:

- ADDVND—Add vendor master record
- CHKVND—Check vendor master record existence
- UPDVND—Update vendor master record
- VALVND—Validate vendor master record

Each function takes and returns different parameters depending on what action is to be performed. The CHKVND function, for instance, uses the following passed fields:

- CMPNO—Company number
- CORPR—Corporation number
- PLTNO—Plant number
- VNDNO—Vendor number
- MODE—Check type (ADD or UPD)

If the mode is ADD, a message will be returned if the vendor already exists. If the mode is UPD, a message will be returned if the vendor does not exist.

The UPDVND function uses a large number of fields, which are checked for individual validity as well as referenced against other files for referential integrity. Among other data passed to the server program are:

- VNAME—Vendor name
- VADD1—Vendor address line 1
- VADD2—Vendor address line 2
- VADD3—Vendor address line 3
- VCONT—Vendor contact name
- VCRCD—Currency code

We expect to map vendor master records to the San Francisco common business objects using the BusinessPartner class with *customer supplier data*. The class structure provided is particularly appropriate and intuitive, because much of the information that is commonly collected for a supplier (vendor) or customer is similar. In addition, it is not unusual for a business partner to fill both roles simultaneously. Basic documentation for the BusinessPartner class definition from the second early driver is shown in Figure 2.

We expect that this class will be usable as provided. For other classes we will extend or subclass when necessary. A common approach used with San Francisco's supplied classes is to use *properties* to add flexibility to a class, so that there will be less need to change, extend, or subclass it. Properties are implemented as a keyed table (hash table), associated with a class, that can be modified without changing the class definition. We are hesitant to use properties, even though they would allow for the fewest changes to the supplied classes. Our concern is that the use of properties could lead to direct manipulation of data associated with (encapsulated in) an object rather than access through the methods of that object. The DescribableDynamicEntity class mentioned in Figure 2 extends the DynamicEntity class, which in turn extends the PropertyContainer class described in Figure 3.

Figure 2 Class definition for BusinessPartner

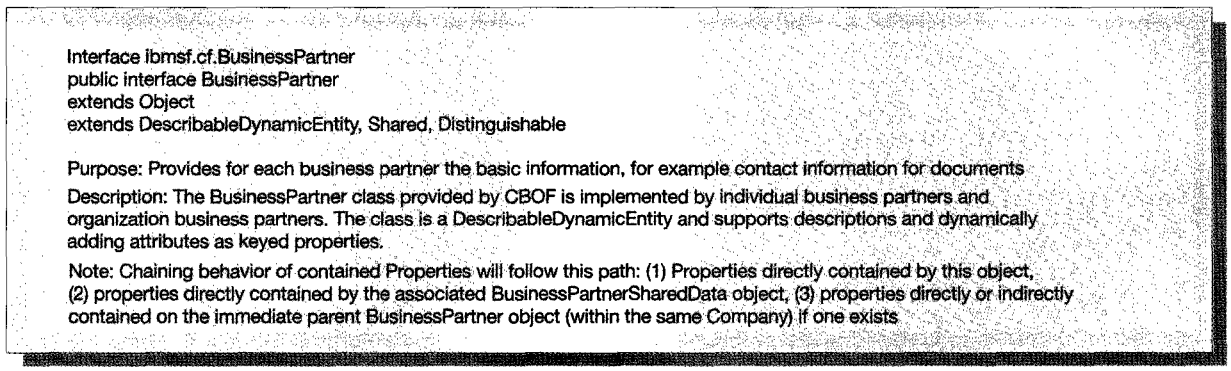
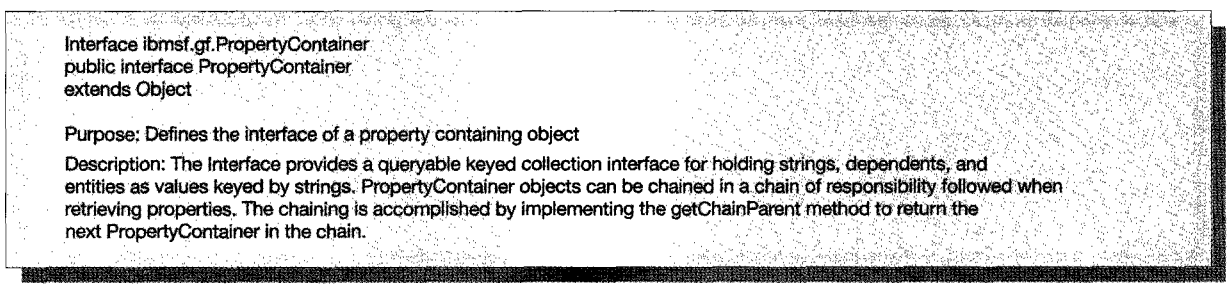


Figure 3 Class definition for PropertyContainer



Although support for a queryable keyed collection is important, direct public access to an object's attributes could lead to misuse. A property container exposes the following methods:

- addDirectlyContainedPropertyBy(Object, String)
  Adds either an unowned Entity, owned Dependent, or owned String property to this PropertyContainer, keyed by the given property key
- addDirectlyOwnedPropertyBy(Entity, String)
  Adds an owned entity to this property container keyed by the given property key
- directlyContainsPropertyKey(String)
  Determines if property key is contained directly within this property container
- getDirectlyContainedPropertyBy(String)
  Retrieves the property found directly within this property container with identifier matching property identifier
- getPropertyBy(String)
  Retrieves the first property found in the chain of responsibility with identifier matching property identifier

Our concern is that adding and getting properties directly by key could foster a tendency to use these methods directly from other objects. Logic used to determine the validity of a property would then be embedded in objects not directly containing that property, but rather in any number of objects communicating with it. We expect that most of our extensions will be the result of subclassing. This will mean adding new variables and methods as well as overriding methods where appropriate, which is more in concert with traditionally accepted OOP techniques.

Numerous methods are supplied with the BusinessPartner class:

- addDirectlyOwnedAddressBy(Address, String)
  Adds an address to the collection of addresses by the specified key
- containsAddress(Address)
  Checks if the address exists in the collection of addresses

- getAddressBy(String)
  Gets the address associated with the specified key
- getBusinessPartnerCustomerData( )
  Returns the business partner customer data, copied from the handle
- getBusinessPartnerSupplierData( )
  Returns the business partner supplier data, copied from the handle
- getDirectlyOwnedAddressBy(String)
  Gets the specified address from those addresses directly owned by this business partner
- getLegalName( )
  Retrieves the legal name for this business partner
- setAddresses(EntityOwningMap)
  Sets the collection of addresses

Many of these methods will easily map from our Vendor module to the BusinessPartner class. The UPDVND function described earlier uses a parameter list with a number of fields, which in turn are checked for validity, and impact on referential integrity could be replaced by a series of method invocations. A preliminary mapping for the VNAME passed field uses the getLegalName( ) and setLegalName methods; for the VCONT passed field the preliminary mapping uses the getAddressBy(String) and setAddresses(EntityOwningMap) methods.

Although getting and setting addresses by strings suggests the use of properties within a business partner, the encapsulation of properties (if this is the implementation) is proper within the BusinessPartner class. Use of these methods does not run counter to our intent not to use properties to enhance classes. It would only be in conflict if we attempted to directly manipulate the addresses.

Future direction. We expect to continue to directly map as many servers as possible to IBM-supplied common business objects. The extent of this, as well as opportunities for new servers, will depend on a thorough investigation of updated descriptions from the final release version of San Francisco as well as other opportunities provided by newer versions of the Java Development Kit. A reexamination of earlier efforts will also be needed because of upgrades to both the JDK and San Francisco. We expect, for instance, that the currency server mentioned earlier may be mapped to the Locale class delivered with JDK 1.1.

New objects outside of the common business objects framework and our current server structure are also being modeled, and we feel that it is important to remain current in Java trends. This has led us to explore JDK 1.2 and its additions to the abstract windows toolkit, as well as Java Electronic Commerce.[17] By using new or extending existing tools, client programs that now access servers may be modified to work with objects. Some of these objects will ultimately replace our servers.

An additional consideration is the frameworks and functions that are now or will be part of the San Francisco project. This is particularly important because of our connection with OAG and holds significant promise both for us and our customers. Interoperability between modules will expand opportunities for ISVs and provide customers with more choices between nonproprietary vendors who are attempting to create the "best-of-breed" solution to a problem. As initially delivered, the San Francisco frameworks emphasize financial interfaces (which is appropriate for a new initiative that is expected to evolve). We expect to use frameworks more extensively as more manufacturing-oriented functions become available.

Since the beginning of our involvement on the San Francisco project, an additional technology (the AS/400 ToolBox for Java[18]) has become available. This allows Java to be used to execute client/server functions via Java classes wrapping client-access API calls. Although we have and will continue to experiment with this approach, we feel that the "100 percent pure Java" initiative has significant value, and we expect a significant return on investment from our involvement with the San Francisco project.

Conclusion

OOP holds the promise of better products deployed by ISVs in less time. Common objects will speed the acceptance of the new technology, and San Francisco adds a new level of ease to integration between ISV- and customer-developed products. The San Francisco project is important in making this paradigm shift an orderly evolution for both ourselves and our customers.

Acknowledgments

The author would like to thank Hank Ternes and Norm Baran for their input on advanced planning and scheduling and Linda Nelson for her insights on Acacia Technologies' server technology; thanks also to Paula Richards and Hal Frye of IBM for their assistance throughout Acacia Technologies' involvement in the San Francisco project.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Sun Microsystems, Inc.

Cited references and notes

1. I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard, *Object-Oriented Software Engineering*, Addison-Wesley Publishing Co., Reading, MA (1992).
2. B. F. Webster, *Pitfalls of Object-Oriented Development*, M&T Books, New York (1995).
3. A. H. Lindsey and P. R. Hoffman, "Bridging Traditional and Object Technologies: Creating Transitional Applications," *IBM Systems Journal* **36**, No. 1, 32–48 (1997).
4. T. E. Potok and M. A. Vouk, "The Effects of the Business Model on Object-Oriented Software Development Productivity," *IBM Systems Journal* **36**, No. 1, 140–161 (1997).
5. D. A. Taylor, "The Use and Abuse of Reuse," *Object Magazine* **6**, No. 2, 16–18 (April 1996).
6. P. G. Basset, "The Paradox of Reuse," *Object Magazine* **6**, No. 2, 58–63 (April 1996).
7. D. A. Taylor, *Object-Oriented Technology: A Manager's Guide*, Addison-Wesley Publishing Co., Reading, MA (1991).
8. According to the *APICS Dictionary*, published by the American Production and Inventory Control Society, Inc. (1987), engineer-to-order products are "products whose customer specifications require unique engineering design or significant customization. Each customer order then results in a unique set of part numbers, bills of material, and routings."
9. According to the *APICS Dictionary*, with mixed-mode manufacturing "the system supports coexistent manufacturing capabilities of repetitive, process, and discrete manufacturing," where *repetitive* means "production of distinct units, planned and executed to a schedule, usually at relatively high speeds and volumes," *process* means production that "adds value by mixing, separating, forming, and/or chemical reactions," and *discrete* means "production of distinct items such as automobiles, appliances, or computers."
10. System APIs (application programming interfaces), user spaces, and the User Interface Manager are available as part of the OS/400™ operating system.
11. References to servers and server protocols are from *Acacia Technologies Application Servers Users' Guide*, Release 8.4 (October 1995); available from Acacia Technologies.
12. The Open Applications Group (OAG) was formed in 1995 as a nonprofit organization dedicated to promoting open applications integration (connectivity and multiple-source integration among enterprise software applications). For more information, see *Open Applications Group Integration Specification*; available from http://www.openapplications.org.
13. In conjunction with our investigation of Java and our involvement with San Francisco, we have used J++™ from Microsoft, Visual Cafe™ from Symantec, JBuilder™ from Borland, and VisualAge™ for Java from IBM, as well as JDK from SunSoft, versions 1.0.2 through 1.1.3.
14. The San Francisco reference group consists of ten software vendors who have collaborated with the San Francisco developers and are early adopters of the frameworks.
15. We did, however, see this as an advantage in the development cycle. We were involved in testing two external drivers (early versions of San Francisco) and were comfortable assuming that development effort expended in working in Microsoft's Windows NT™ would be transferable to an AS/400 platform when the Java Virtual Machine was available there.
16. This figure is adapted from a presentation provided by Paula Richards of IBM, when she was project manager for the San Francisco project.
17. For more information, see http://www.sun.com/products/commerce/jecf_arch_intro.html.
18. For more information, see http://www.as400.ibm.com/products/software/javatool/javabeta.htm.

Accepted for publication January 8, 1998.

Richard A. Henders *Acacia Technologies, Computer Associates International, Inc., 2400 Cabot Drive, Lisle, Illinois 60532 (electronic mail: henri03@mail.cai.com).* Mr. Henders is currently a software specialist with Acacia Technologies. He began his data processing career with IBM in Chicago, after completing service as an artillery officer in the United States Marine Corps during the Vietnam War. He received a B.S. degree, with majors in both mathematics and American literature, from Blackburn College, Carlenville, Illinois, in 1967. A member of ACM and IEEE, Mr. Henders has held a variety of software development and management positions in private industry, consulting firms, and software development firms.

Reprint Order No. G321-5671.