IBM Business Frameworks: San Francisco project technical overview

Software developers face significant challenges as they attempt to modernize current applications to take advantage of the benefits of distributed objects. The cost of developing the next generation of applications as industrial-strength distributed-object solutions will be prohibitive for many software vendors, because their skills are in developing procedural applications. Some vendors have indicated that as much as 80 percent of their development cost is in writing and supporting the basic, noncompetitive functions that are essentially the same for any application solution offered in a specific domain.

The San Francisco project now in progress addresses these problems by providing object-oriented infrastructure and application logic that can be expanded and enhanced by developers in the areas where they choose to provide competitive differentiation. The frameworks are intended to lower the barriers to widespread commercial implementation of distributed object solutions. This report provides an overview of the IBM San Francisco project and its Business Frameworks.

The San Francisco project was started when several software vendors asked us for help in modernizing their application products. They realized that their current applications needed to be updated if they were to continue to be viable in the emerging object-oriented, network-based market. However, there were several barriers. One barrier was the problem of retraining the development staff to effectively use object-oriented technology. The retraining would be more than just learning another programming language; the staff would need to learn how to analyze a problem in terms of objects and how to use that analysis to design an object-oriented solution. A whole new approach to building systems and a whole new set of skills and tools would be required.

A second barrier was the risk involved in moving to a new technology. Often the first solution built by a team using new skills and a new technology is less than perfect. A poor design causes problems, such as code that does not function properly, poor performance, or a solution that is hard to use. Solving such problems is a necessary step in learning to apply a new technology, but the number and magnitude of the problems must be contained, so that the business can keep operating while the new approach is learned.

A third barrier in moving to object-oriented technology was the cost of making the change. The software developers needed some basic infrastructure upon which to base their applications. Many of the companies could not afford to develop this infrastructure themselves. They also could not afford to rewrite their entire product line at one time. They needed to be able to spread the cost of upgrading their applications over time by having the object-based portions of the application interoperate with portions that had not yet been updated.

The San Francisco project helps developers to overcome the barriers through *business frameworks* that provide an object-oriented infrastructure, a consistent application programming model, and some default business logic. The frameworks make it easier to move to object-oriented technology because developers use well-tested services instead of building their own. They can design their solutions using a proven programming model instead of developing a unique approach. They can build their applications by modifying and extending the default business objects and logic provided in the frameworks instead of having to start "from scratch" to build applications from raw requirements statements. This allows developers to apply more development resources on the functions that will give them a competitive advantage. The frameworks are designed to be easy to extend in areas where software vendors have told us they differ from their competitors.

The San Francisco project is being developed in collaboration with several hundred international ISVs (independent software vendors) to ensure applicability across a broad range of small-to-medium-enterprise business solutions. The ISVs are working with IBM to design, develop, and validate frameworks, create development tools, and develop integrated applications using the frameworks.

Feedback from companies that have tested early versions indicates that the frameworks provide about 40 percent of a typical working application within the supported domains. ISVs will develop the remaining 60 percent of the application on top of the frameworks and bundle both the IBM and ISV code into a single solution, which the ISV will then license to customers. The customers benefit through improved flexibility to meet changing business needs, improved availability and affordability of customized multiplatform business solutions, and improved application interoperability. Use of a shared architecture will make it easier to integrate solutions from different software vendors.

Frameworks description

The San Francisco project is building three layers of extensible components for use by application developers. In the highest layer, the core business processes provide business objects and default business logic for selected "vertical" domains. The second layer provides definitions of commonly used business objects that can be used as the foundation for interoperability between applications. In the lowest layer, the base provides the infrastructure and services that are required to build industrial-strength applications in distributed, managed-object, multiplatform applications. The base isolates an application from the complexities of multiplatform network technology and allows the application providers to focus on unique elements that give value to their customers.

Application developers may choose to use the framework technology for only portions of their application. The San Francisco frameworks have been designed to coexist with existing business applications, preserving existing application investments.

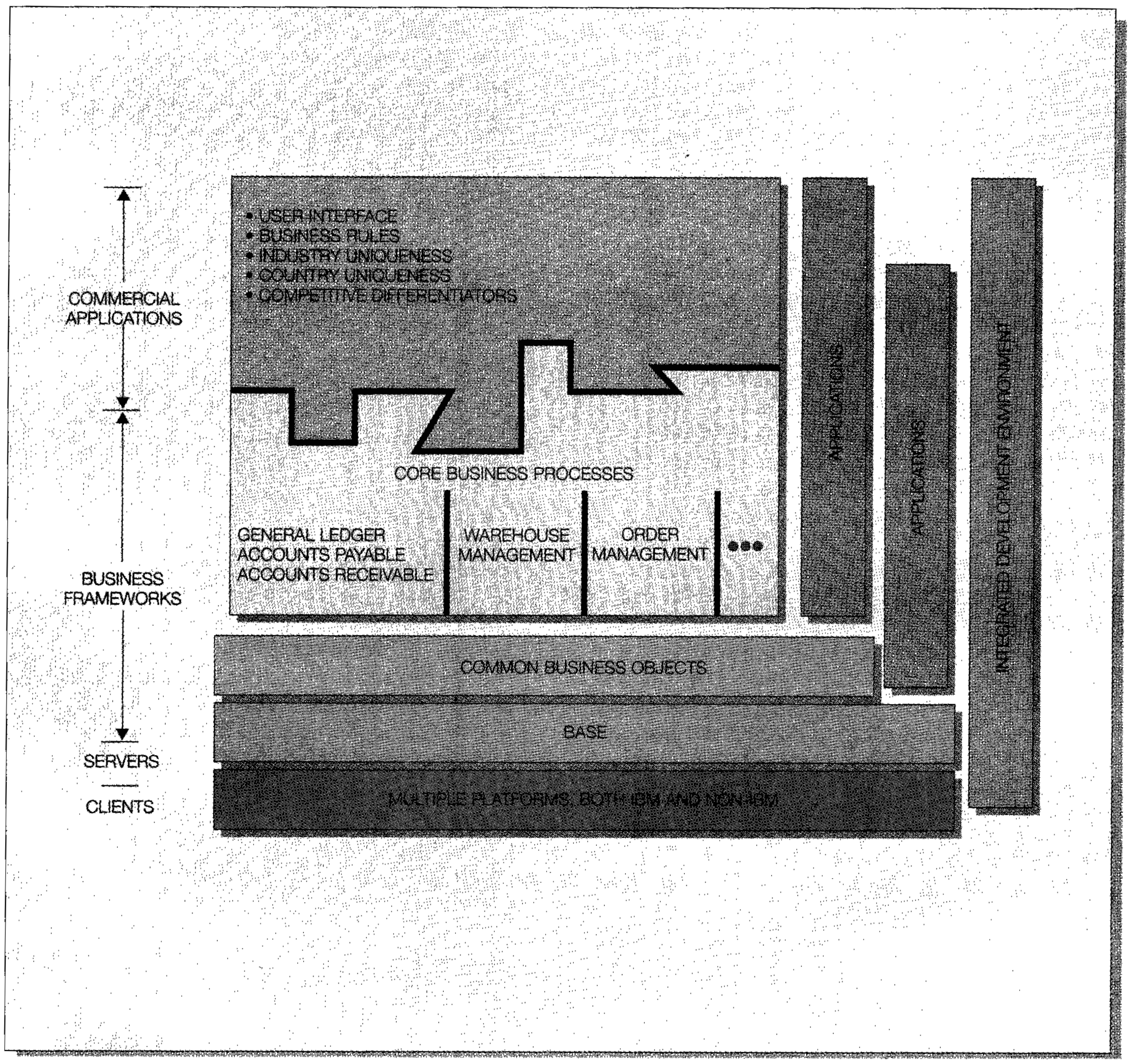
Application developers can use the frameworks at any of the three levels, as shown in Figure 1. At the lowest level, application developers can utilize the base infrastructure to provide a consistent programming interface and structure for building distributed multiplatform applications. At the next level they can select common business objects as the basis for application integration. This level provides a common foundation for building interoperable business solutions. At the highest level, application-specific business frameworks will provide core business processes that can be easily extended to provide a complete business solution. Initially, the San Francisco project has examined business application frameworks in the domains of accounts receivable, accounts payable, general ledger, sales and purchase order management, and warehouse management. Over time, these business frameworks will be extended and enhanced with additional objects and access to more framework interfaces, providing greater application flexibility.

The San Francisco project (referred to simply as "San Francisco" in the rest of this report) uses the Java** language. This makes the frameworks, and the applications developed using them, portable across many platforms. It also allows developers to use the many tools and class libraries the industry is producing for Java development. We expect most development with San Francisco frameworks to take the form of Java applications. However, application developers can create applets[1] to work with San Francisco objects as well.

The following information reflects the current direction for the San Francisco frameworks. IBM has not committed to ship all of the functions and reserves the right to make changes to functions listed here. We are working with our reference group[2] to help validate and prioritize how functions will be delivered over time. Our current plan is to ship the base technology, several common business objects, and portions of the the general finance vertical domain frameworks in the initial release.

Base layer. The lowest-level framework that can be used by application developers is the base. It provides the underlying infrastructure for the common business objects and the core business processes. It allows San Francisco to hide differences in underlying technology from application developers, making it easier to support an application on multiple platforms while still exploiting platform-specific advantages. The base layer also provides a consistent programming interface and structure for building applications.

Figure 1 IBM Business Frameworks



Two categories of functions in the base layer are directly usable by developers: *base object model classes* and *utilities*. To support distributed, mission-critical requirements, the base layer also provides a set of *kernel services*. In most cases, the kernel services are not directly visible to developers. Instead they are invoked indirectly by the base object model interfaces. This approach helps to simplify the application programming model. It also will allow application developers to make use of new technology that IBM might incorporate into the infrastructure without modifying their application code. The interfaces that application developers use would remain consistent; only the underlying implementation of the infrastructure would change.

Many of the services in this layer are based on object service definitions from the Object Management Group (OMG). For example, the kernel service provides an object transaction service, collection handling, communication between distributed objects, and persistence management. However, San Francisco is not providing a CORBA**-compliant Object Request Broker.[3] The base merges and combines OMG-defined functions with functions provided by Java. It also simplifies the OMG definitions when possible and adds additional function when necessary. For example, instead of supporting all of the locking types defined by OMG, we are supporting only optimistic locking (to allow greater concurrent access to data) in addition to the traditional pessimistic locking approaches used by many infrastructures. Our services are also influenced by other sources, including Taligent frameworks[4] and patterns as described by Gamma et al.[5]

The kernel services contain extensions that we found were necessary for our Java frameworks. We are using Java's remote method invocation (RMI) interface as the basis for the communication infrastructure. We have extended the RMI function to support areas such as server process management. We anticipate that some kernel services that complement the base will be provided by products from other vendors. Examples include licensing and encryption technology.

Object model classes. The object model classes provide the basic structure for San Francisco objects and frameworks. In effect, they define the San Francisco application model. These classes contain complete methods that are inherited and used by application developers, as well as abstract definitions for other methods.

The object model classes provide a consistent object model that will fit a wide range of distributed applications. They allow the developers to specify different options, such as defaults for location (on the client or on the server), locking mechanism, and object identification approach. The interfaces may be implemented differently to exploit the advantages of each platform, but the behavior seen by the developer is the same.

The object model classes include:

- Entity. Entities are independent, shareable objects (persons, things) that are used in the operation of a business. Entities are often associated with the data that are at the core of an application or framework. An entity may be persistent (associated with an underlying persistent storage mechanism on a server) or transient. Entities may also be used for temporary caching of data on client systems while processing is taking place. The methods associated with each entity generally deal with getting and setting the state (attribute) values, or business logic that involves only a single object. A subtype of Entity, Dynamic Entity, allows property/value pairs to be associated with the entity at execution time. This provides great flexibility in customizing entities to specific business requirements.
- Dependent. Dependent objects have less system overhead than entities. Because they cannot exist outside of the scope of an entity, they cannot be shared, be referenced, or take part in transactions independently. Dependents often contain additional information about their owning entity.
- Command. A command is a group of operations on an object or collection of objects. Commands contain logic that is applicable to more than one business object. The commands contain many of the functions and procedures for an application or framework. They can be distributed to either clients or servers and can affect either individual entities or collections of entities.
- Collection/Iterator. Collections are used to group objects together. Some collections are structured so that individual elements may be accessed by a key. Others function as a set of elements. Iterators associated with the collection are used to access the elements and traverse across the collection. For example, an iterator may provide a "next" method to enable scrolling through a collection.
- Factory. A factory manages instances of its objects during framework execution. Factories provide functions that create and delete entities, commands, and collections. Different implementations of the factories will allow application developers to support different platforms and persistent storage mechanisms with minimal or no change to the business objects themselves.

Utilities. Utilities provide services that will be needed by most applications built from the San Francisco frameworks. The utilities are designed to be used "as is," rather than extended or modified.

Several different types of utilities are provided. *Administration* supports the definition and maintenance of *application security* and *system configuration* information. *Conflict control* allows a system administrator to prevent commands that should not be executed at the same time from running concurrently. The *installation* utility helps application developers install and maintain the frameworks, and applications built from the frameworks. The *audit trail* utility allows application developers to track object access by users.

Many of the utilities will be implemented by invoking functions provided in the operating systems or in other products. The San Francisco developers will ensure that the functions are well-integrated and provide a consistent "look-and-feel" for users. And, like the kernel services, the utilities will provide additional capability when needed. For example, in application security the utility adds support for grouping users by the methods they can access.

Common business objects layer. The middle layer of the frameworks contains the *common business objects* (CBOs). This layer is composed of several independent frameworks that can be categorized as (1) business objects common to multiple domains or (2) common application services. Speaking in general terms, the business objects represent those entities that a person knowledgeable in the domain would reference when describing how to perform a business task in nontechnical terms. The common application services are more likely to be identified when discussing an approach for automating a process.

Business Partner is an example of a business object that we found in multiple domains. It encapsulates the characteristics of a customer or supplier, such as the default currency, a description, and the languages used by the partner. Another example is *Address*, which provides a generic way to describe a location, including a postal area. It supports different formatting controls for the address data and the relationships of an address to other objects, such as locale and language. An example of a common application service is the *Decimal Structure*. It provides the capability to define the number of decimal positions and rules about how the number is to be processed, such as rounding on input or output.

For many of the common business objects, part of the basic structure and behavior is required by multiple application domains and part is unique to an individual domain. For example, in the Business Partner, much of the structure related to currencies, languages, and addresses would be required by multiple domains, while the structure and behavior associated with the product-supplier relationship would be unique to the warehouse logistics domain. In this case the common portions are implemented within the CBO layer and the unique structure and behavior is implemented as part of the application domain framework that references the CBO structures.

Core business processes layer. The top layer of the frameworks contains the *core business processes*. The objective for this layer is to create a sound architecture and highly extensible objected-oriented implementation for the basic structure and behavior of any solution in the selected domain. On top of this basic structure and behavior we will implement a very limited set of application functionality, so that the frameworks will actually do something as they "come out of the box." Our participating vendors tell us that the combination of common business objects and core business processes will approximate 40 percent of a typical working application. It is anticipated that application providers will in all cases extend the frameworks to add their own user interface, country- and industry-specific requirements, business rules, competitive differentiators, and complementary application functions. The extension points at which application providers will add or replace business logic are carefully defined during the frameworks design phase.

The application domains addressed in the initial requirements and design phases for the San Francisco project included business "financials," (accounts payable, accounts receivable, and general ledger), order management (sales orders and purchase orders), and warehouse management (logistics and control functions). The initial toolkit for San Francisco contains the General Ledger framework, several common business objects, and the base infrastructure. Additional frameworks and business objects will be added over time, based on customer requirements.

The frameworks in each domain make use of common business objects and provide the structure and default behavior for relevant business tasks. Examples of tasks in the Accounts Receivable/Accounts Payable Ledger framework are Payment (receiving payments from and creating payments to business partners) and Transfer Item (transferring an item from one account to another, or from one business partner to another).

The General Ledger framework can be used to manage the accounts on the general ledger for a company or a hierarchy of companies. Business tasks supported include Journaling (creating, validating, processing, and posting journals) and Closing (closing the books for an accounting period or year).

The Sales Order Processing framework manages quotations, sales orders, and sales-order contracts throughout their respective life cycles. Its business tasks include Pricing and Discounts (maintaining, retrieving, and calculating sales prices and discounts) and Sales Contracts (creating and maintaining sales contracts and tracking customer compliance).

The Purchase Order Processing framework supports purchase orders and supply contracts. Its business tasks include Purchase Orders (creating, maintaining, and confirming purchase orders) and Back to Back Orders (managing purchase orders that are directly linked to specific sales orders).

The Order Management framework provides an abstract model and default behavior for aspects of order processing that are common across several order-related processes (e.g., sales orders, purchase orders, and quotations). Within this framework Order Data Interchange is an abstract model for managing order data that are interchanged among several entities. It includes preprocessing to select and normalize data before the data reach the actual processing destination.

The Warehouse Management framework supports warehouse logistics tasks, for example, Internal Replenishment (recommendations for stock movements between warehouses), and Kit Assembly (tracking the associated stock activity and movements). A business object used in this framework is the Product (definition, policies, lead time, reserves, and balances). Other tasks supported include Manual Stock Transactions (receiving and disbursing stock for miscellaneous purposes) and Kit Definition (defining a kitting operation to assemble the product).

Building applications from frameworks

Building applications from the business frameworks will be approached in several ways. The simplest of these is to use the objects and classes in the frameworks without changing them. To do this, the developer writes client code that uses a factory object to manage access to the framework business objects. The factory manages a command (and the associated transaction) that creates, deletes, or updates a business object.

A second approach is to modify the frameworks by creating new domain classes from the base object model classes. This would be done if a new business task or process is added on top of the frameworks. Developers who make these changes need to understand the methods and programming guidelines for creating, deleting, and updating framework objects. They will need to understand how to use the factory methods that create and delete business objects and how to build the commands that implement the new business task.

A third approach is to modify the frameworks by extending the supplied domain classes and methods. It may be necessary to add additional attributes to those defined for a class, or to replace the logic in one of the methods. We worked closely with our participating vendors to identify the types of changes that will be necessary. We then designed the frameworks so that these changes are easy to make and affect other parts of the frameworks as little as possible. Developers must understand and follow the application model of the base object classes to ensure consistency with unchanged parts of the frameworks.

Example. The San Francisco frameworks are designed to be extended easily by application developers. Extensions include overriding the default business logic in supplied methods, adding attributes to existing classes, adding methods to existing classes, and adding new classes to the frameworks. The framework documentation will describe each part and its function, helping the developer to find the section of code to be customized.

As an example, consider a framework that includes the classes Receipt and Purchase Order Line. Default attributes of Receipt include status, quantity accepted, and quantity rejected. Its default methods include determining if an inspection is required and recording inspection results.

A default attribute of Purchase Order Line is a "quality inspect" flag that can be set to "yes" or "no." Receipt's default method to determine if an inspection is required simply tests the value of this attribute.

Suppose the developer wants to add logic to make additional tests before determining if an inspection is required. Perhaps information about the supplier, or previous receipts from the supplier, should be checked. Perhaps hazardous materials and expensive products should always be inspected.

To do this, the developer creates a subclass of Receipt and writes a new method to determine if an inspection is required. This method overrides the default method and contains the logic needed to check the supplier table, check the results of previous receipts from this supplier, and determine whether the receipt is for hazardous materials or high-value products before making the final determination that an inspection is or is not required.

When the changes have been compiled and installed, the new logic will be in effect for quality checks, while the rest of the framework continues to function as before. Of course, if the new method requires classes or attributes that are not already in the framework, additional changes are needed.

Development process

Effective use of frameworks requires a different development approach than is typically used in developing applications. A development process tailored for frameworks is outlined here:

1. Business architecture. First, domain experts define the business problem to be solved. This involves gathering requirements from experts, users, and existing systems. The business problem is broken down into business processes. These are viewed as functional requirements, or use cases.[6] The framework repository is searched for available processes that match, or can be extended to match, the functional requirements. The required processes are assembled in an object model diagram[7] to analyze static elements of the design. The use cases are used, in conjunction with the object model, to define object interaction diagrams[6] to analyze the dynamic aspects of the design.

2. Framework componentry. The focus now moves to finding or defining reusable parts. The repository is browsed to find parts that meet or can be extended to meet the functional requirements incorporated into the model. In some cases the function needed may already exist in the repository. In other cases developers may need to use combinations of existing frameworks or modify existing frameworks to meet specific needs. This may include adding and deleting activities and nodes, or modifying the sequences of activities. If the function does not exist in the repository, one may consider buying a framework to incorporate into the design. In still other cases new code will need to be developed. It is important that it be designed using object techniques to maximize its potential for reuse.

3. Application architecture. Developers are now ready to build the application. This includes incorporating the user interface with the functions needed to meet all design requirements. The application is assembled by pulling together the selected frameworks and developing any additional code that is needed. The application is prototyped and tested. The frameworks are changed and extended as required to meet application needs. There should be two to three complete design cycles, from analysis through prototyping. This is critical to ensure that the basic design is right. It will become the foundation for defining and testing incremental improvements to the application. Following this step, the application is ready for end-user testing.

4. End-user environment. The application is installed and integrated into the business environment. For simple applications, the customer may simply install and configure the shrink-wrapped product. A complex application might be validated via process modeling and analysis, tested with end users, and integrated with other applications. New requirements might be incorporated before the application is finally put in production.

Tools are needed to support each step of the process. For San Francisco, tool support will be provided by both IBM and other vendors. We are working with tool vendors to ensure that their tools make effective use of the frameworks.

Development approaches

The ultimate goal for the San Francisco frameworks is to support rapid application development. The developers will be coming from several different application disciplines.

San Francisco will allow the integration of different approaches for building applications, for example: compound documents, business process modeling and control (workflow), and Java-based internet/intranet approaches. It is our plan to provide sample demonstrations of how these technologies can be integrated.

Compound documents. Compound document environments include Lotus Notes** and scripting, JavaBeans**, and ActiveX**, (all supporting desktop-centric, document-style applications. Development in these environments focuses on how a document is presented to the end user and how the information is represented. Attached to the various sections or parts of the document form are scripts containing the functional extensions.

For example, a form might represent a sales order. As the sales order passes through its life cycle, scripts are executed and data values are retrieved and placed in the form: name and address in the heading information, inventory items in the body, and calculated totals and discounts in the totals area.

San Francisco frameworks can be used as a basis for the form and its parts, providing the functional extensions for these compound documents.

Business process modeling and control. Business process reengineering is giving focus to a number of technologies, including modeling tools and workflow process-control engines. Workflow technology provides a layer of control outside of the normal processing. The outer layer can be modified to meet business process-to-market demands, while keeping controls and audits for existing processes. This supports a customer focus on business processes, to streamline and control them for increased efficiency and cost savings.

Application developers in this environment can use San Francisco frameworks as business activities within a process, and also to initiate business processes. They will use workflow products to "glue" various new and existing applications together into a coherent business process, passing parameters between application activities via workflow products.

Internet/intranet and Java. This is one of the fastest-growing and most exciting technologies for developing distributed application solutions. Applications may be designed and configured to either include execution of objects on the client or to have all execution on the server. Solutions will need to allow seamless integration between Java and non-Java environments.

Migration and coexistence

Businesses will tend to upgrade and replace their mission-critical applications a few at a time rather than all at once. San Francisco will provide ways for new applications based on the frameworks to coexist with existing business systems.

First, the frameworks will allow integration with legacy databases. San Francisco will provide "schema mappers" to allow developers to access relational databases as San Francisco objects. These databases may be part of a new application design or from existing applications. An open interface will support relational databases from several vendors. Our goal is to allow shared, concurrent update access between legacy and San Francisco-based applications.

A second area to address is the interoperation of legacy code with San Francisco objects. We plan to demonstrate that San Francisco objects can call functions provided by applications written in traditional languages. This provides access to a large set of legacy application function. The ability for other programs and objects (such as ActiveX objects) to invoke San Francisco objects is also being examined. Many groups, both within and outside of IBM are working on this problem. We plan to provide sample code to help ISVs better understand the interoperability requirements for migration and coexistence.

Another important consideration in migrating to San Francisco-based applications is education. Developers must learn how to design and develop objects. They must understand what functions the San Francisco frameworks provide, and how to extend and customize them for their own applications. IBM will provide training on how to build frameworks and frameworks-based applications.

Standards activity

There is ongoing work in the industry to develop standards in the areas of Java, business objects, and business object infrastructure. One area being explored is the relationship between Java and object request brokers (ORBs). Another is the type of infrastructure required to support business objects in distributed computing environments. There are efforts to reach agreement on common definitions for some business objects.

The San Francisco project is participating in many of these discussions. We are working, along with other projects in IBM, to submit proposals to the OMG in the areas of business object facilities and common business objects. Standards in these areas will allow customers to more easily combine software from several vendors into an integrated solution. Standards will also make it easier for application developers to learn to use distributed objects. However, the software vendors working with us ask that we not wait for all of the standards questions to be answered. We are proceeding to build and deploy the frameworks while contributing to the definition of standards.

Early vendor involvement

The San Francisco project was initiated because of requests from software vendors to help them find a way to take advantage of object-oriented technology. These vendors continue to be involved, reviewing our plans and designs at scheduled "advisory group" meetings. Their contributions and guidance are essential to ensure that the frameworks will help them as they build their future products. More information on becoming involved in the San Francisco project can be found at our frameworks home page: http://www.ibm.com/java/sanfrancisco.

A pervasive question from our advisory group is "what should we do to prepare our development organizations to use object-oriented technology and the San Francisco frameworks?" In answer to this question we are offering a number of education courses. Introductory courses cover basic object-oriented concepts and the Java programming language. Additional courses are available on object-oriented analysis and design, and on how to select initial projects that will have a high likelihood of succeeding. Finally, courses are given on the San Francisco frameworks, and how to use them as the basis for designing and building frameworks-based applications.

Summary

The San Francisco project is intended to help application developers rapidly build distributed, object-oriented applications. It provides a base of object-oriented infrastructure and application logic, which can be expanded and enhanced by each developer. This report has given an overview of the San Francisco project and the IBM Business Frameworks being developed. It is intended to provide the reader with a high-level understanding of the architecture and content of the frameworks.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Sun Microsystems, Inc., Object Management Group, Lotus Development Corporation, or Microsoft Corporation.

Cited references and notes

1. A Java applet is automatically downloaded, when invoked through a browser over a network, and runs on the client's machine.
2. Our reference group consists of ten software vendors who have collaborated with the San Francisco developers and are early adopters of the frameworks.
3. Standards have been published by the OMG for the Common Object Request Broker Architecture (CORBA). More information about the OMG can be found at http://www.omg.org .
4. *Leveraging Object-Oriented Frameworks: A Technology Primer from Taligent,* Taligent, Inc. (1993); available at http://www.taligent.com/Technology/WhitePapers/LeveragingFwks/LeveragingFrameworks.html .
5. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Publishing Co., Reading, MA (1995).
6. I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard, *Object-Oriented Software Engineering, A Use Case Driven Approach*, Addison-Wesley Publishing Co., New York (1992).
7. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ (1991).

General references

L. Lemay and C. Perkins, *Teach Yourself Java in 21 Days*, Sams.net Publishing, Indianapolis, IN (1996).

D. A. Taylor, *Business Engineering with Object Technology*, John Wiley & Sons, Inc., New York (1995).

Vincent D. Arnold, Rebecca J. Bosch, Eugene F. Dumstorff, Paula J. Helfrich, Timothy C. Hung, Verlyn M. Johnson, Ronald F. Persik, and Paul D. Whidden

IBM AS/400 Division
Rochester
Minnesota