A Case Study:

Distributed Object Technology
at
Wells Fargo Bank

A Cushing Group White Paper

Michael L. Ronayne and Erik S. Townsend

October, 1996

THE CUSHING GROUP

Business Solutions based on Distributed Object Technology

Table of Contents

Executive Summary

It was once simple to understand the difference between a bank, a brokerage house, a mutual fund company, and an insurance company. They offered fundamentally different products, and there was little overlap. Today, the differences have become blurred, as financial institutions of all types compete for investors' funds. Leadership in the financial services industry requires that an organization be able to quickly adapt to a changing market, and to be timely and efficient in offering new products and services to its customers. Financial services is, inherently, an information-intensive industry, and therefore, an organization's agility in transforming its information systems to meet changing business needs becomes a critical competitive success factor.

Wells Fargo Bank has become a technology leader in banking, thanks in part to the company's pioneering use of distributed object computing technology. This technology has helped the company to achieve several business objectives including changing the way that bank employees work with computers to become customer-focused rather than account focused. Innovations made possible through the use of this technology have enabled Wells Fargo to become the first major bank to offer its customers secure on-line access to account balances through the Internet's World-Wide Web.

Distributed Object Computing technology offers a means of interfacing disparate computer systems with one another, emphasizing the creation of re-usable software components, or *business objects*, which can be combined in a variety of ways to meet changing business requirements. The technology itself is just an enabling factor, however. Wells Fargo's success is owed primarily to the company's willingness to commit itself to technology leadership, and to re-shape part of its IT organization to focus on creation of re-usable software components. Wells Fargo has developed management techniques for structuring an organization to benefit from this technology which serve as a reference model for other companies using the technology.

Wells Fargo has been using distributed object technology since 1993, and has, to the best knowledge of the authors, become the most successful user of this technology. Wells Fargo has put in place large-scale production systems based entirely on communications tools, or *middleware*, which comply with *CORBA*, the industry standard specification for distributed object computing published by the *Object Management Group*, a consortium of over 600 companies.

Distributed Object Computing and the *CORBA* standard were relatively unknown in 1993 when Wells Fargo began using the technology. Recently, both have been widely heralded as major direction-setting factors in the software industry. This is evidenced, in part, by Netscape's recent debut of its "ONE" architecture, which portrays *CORBA* as a key component in the evolution of Internet/Intranet technologies to widespread use in the large business enterprise. Wells Fargo's early adoption of this leading-edge technology serves as further evidence of the company's commitment to technology leadership in banking.

Wells Fargo now bases most development of software systems which support direct, electronic customer access to account information upon an *object model* which relates the bank's business processes, products, and services to a set of re-usable, modular software components. These software components, or *business objects* can be accessed and shared by several application systems through the company's internal TCP/IP network.

This paper chronicles Wells Fargo's experience with this technology, and discusses some of the techniques which have helped enable their success as a technology leader in banking.

Background and Motivation for Change

In the 1980s, competition for investor's funds traditionally kept in banks expanded to include brokerage and insurance companies. To remain competitive, banks began offering products such as mutual funds and brokerage accounts – products historically outside the purview of traditional banking. By the late 1980's, a growing trend was *compound statement banking*, meaning that customers expected to receive a single, unified statement listing the balances and transactions for all of their accounts - ranging from checking and savings accounts to mortgages to credit cards to brokerage and retirement accounts. Wells Fargo realized that to meet customer demand, the bank needed to change from an *account* focus to a *customer* focus. If successful, the customer's relationship with the bank would become more complex with time, and would likely involve several different accounts. This would result in customers expecting their banking activities to be structured in terms of their overall *relationship* with the bank, rather than in terms of a single account they might own.

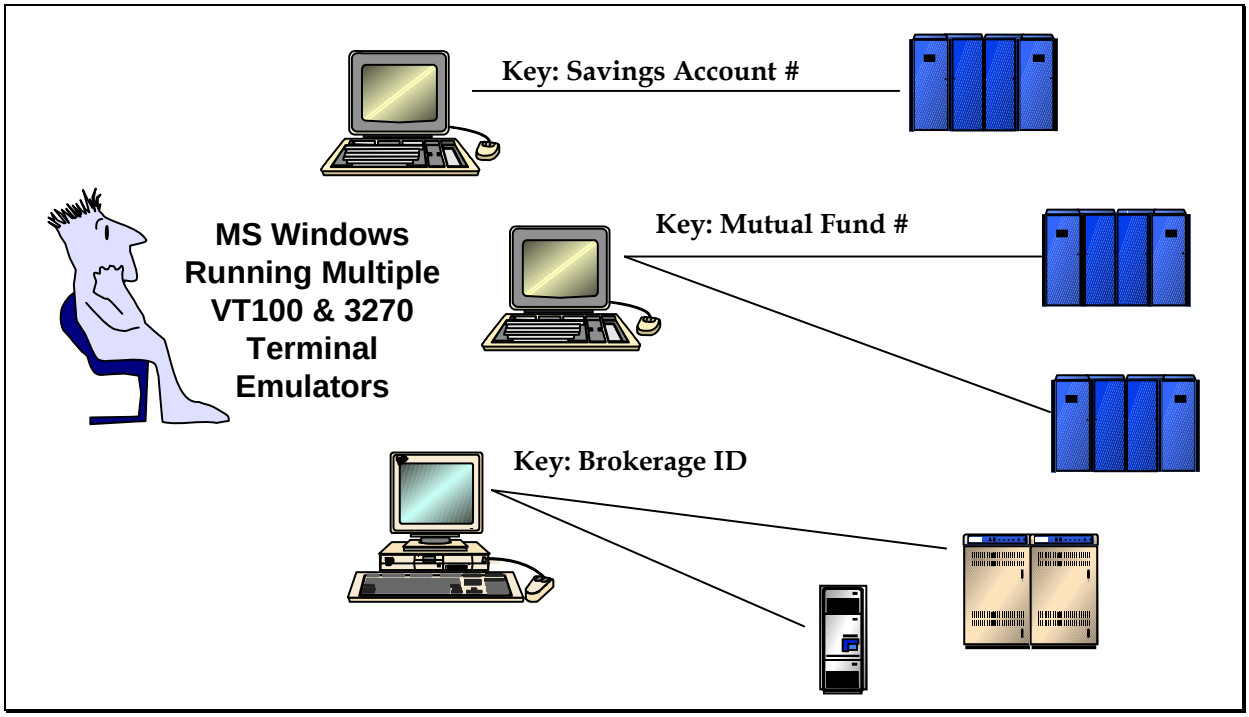


Figure 1 - Information Access Before

Like most banks, however, Wells Fargo's computer systems were optimized for account processing rather than providing integrated customer data. Direct deposit accounts were managed by mainframe computers while mutual funds were tracked by a Digital VAX/VMS system. Meanwhile, brokerage account processing was outsourced to a vendor who used a Tandem system. As a result, bank employees were required to log into several different computer systems, all with character-cell user interfaces. What should have been routine transactions for customers required bank employees to navigate three or more disparate information systems. A simple question from a customer such as *"How much money do I have in all of my accounts combined?" could require* an employee go to several separate different *systems of record*, none of which offered a graphical user interface. This became an impediment to efficient customer service.

Wells Fargo began to seek out remedies to this problem. It was clear that re-engineering all of the bank's systems of record to embrace client/server (or any other) technology would neither be practical nor cost-

effective. Too much investment already existed in the *legacy systems*. Wells Fargo needed a way to overcome the inherent disparity between the existing systems. They needed to seamlessly integrate these systems' functionality, and preferably deliver the integrated result through a graphical user interface. Wells Fargo began to evaluate approaches which could present a graphical interface which integrated data from several systems of record at a practical cost. A key objective was to deliver the integrated functionality through a *native* Microsoft Windows™ user interface.

Wells Fargo began working with several companies including Digital Equipment Corp.'s San Francisco office. The parties discussed the concept of an integrated, PC-based application to help users overcome the difficulties discussed earlier. Digital worked with Wells Fargo to refine the functional requirements for such an application, and relied on engineering groups at Digital's headquarters organization on the east coast to help determine what technologies could be used to meet Wells Fargo's needs.

An engineering group at Digital's headquarters was familiar with The Cushing Group, and referred the authors to Digital's San Francisco office (the authors had previously worked with Digital to help design the company's *ACA Services* product, the world's first commercial *object request broker)*. Digital's San Francisco office in turn introduced us to Wells Fargo.

The Cushing Group first met in October 1993 with Eric Castain, Vice President and Manager of Wells Direct Systems at Wells Fargo, who had just taken on responsibility for finding a solution to the problems discussed earlier. The target users would be the customer service agents who support the "mass-affluent" market – that is, those customers who typically have several accounts and who typically have significant financial assets.

Prior to our arrival, some initial discussions between Digital, Wells Fargo, and The Cushing Group had occurred to roughly define the application's function. Essentially no technical design work had begun before we traveled to San Francisco, however.

Mr. Castain was not one to waste time, and in the course of a full-day meeting we managed to scope and define a 3-month project to provide the telephone agents with a more integrated, intuitive, and easier-to-use interface to customer account information. As part of this project we confirmed the feasibility of using a three-tier client/server architecture to deliver a rapid solution that was production-grade quality. Two options for the client and server relationship management were considered. The choices were DCE or CORBA. While both options were early in their life cycle, the primary consideration was software availability and stability. The long term technical direction was also very important. The Wells Fargo and Digital/Cushing Group teams recommended the use of Digital's Object Request Brokering (ORB) technology[1] to meet these goals. The software was found to be stable and met Wells Fargo's needs. Object request brokering offers a way to construct distributed application systems using object-oriented semantics to define an application-level communication protocol between client and server programs.

Although the Digital product was originally based on a proprietary design, Digital and several other companies had already joined forces to define a standard for object request brokering technology called CORBA.[2] The Digital ORB product was already moving toward the then loosely defined CORBA standard. The selection of Digital's ORB set the direction for the effort within Wells Fargo that would evolve to become what is now almost certainly the world's largest CORBA-based application development

[1] The team recommended Digital's *ACA Services* Object Request Broker, or "ORB" for the initial project. ACA Services was the world's first commercial object request broker, and was based on Digital's proprietary design. Subsequent to the standardization of ORB technology, Digital renamed that product *ObjectBroker*™ when it became fully CORBA-compliant. ObjectBroker™ is the product currently in use at Wells Fargo.

[2] CORBA is an acronym for *Common Object Request Broker Architecture*, and is published by the Object Management Group, a consortia which now includes over 600 member companies. Digital has since re-engineered ACA Services to be fully CORBA-compliant, and has re-named the product ObjectBroker™).

effort[3]. This technology would later enable Wells Fargo to become the first bank to offer Internet-based access to account balances, not to mention several other innovations. The rest of this white paper chronicles Wells Fargo's experience with distributed object technology, beginning with the aforementioned application.

The Customer Relationship System

A Better Model for Customer Service

Mr. Castain shared with us his vision for a solution to the customer service agents' requirements based on the description of the problem he was given by the Telephone Banking group. The discussion began with the assertion that we should "Forget about account numbers as the key focus in this effort". As anyone familiar with banking systems knows, account numbers are the primary index keys for just about everything, so this was a bit unsettling at first. Mr. Castain explained that when a customer who owns several accounts calls the bank, they should not be required to know their account numbers. The customer may wish to do business based on an account number or they may want to transact business based on their overall *relationship* with the bank, which might involve several different accounts, and perhaps, transfers between them. The most fundamental thing missing from the existing systems was an ability for the agent to specify the identity of a customer (Social security number, name, etc.) and then see a complete view of that customer's overall relationship with the bank - all accounts owned, their balances and status, etc.

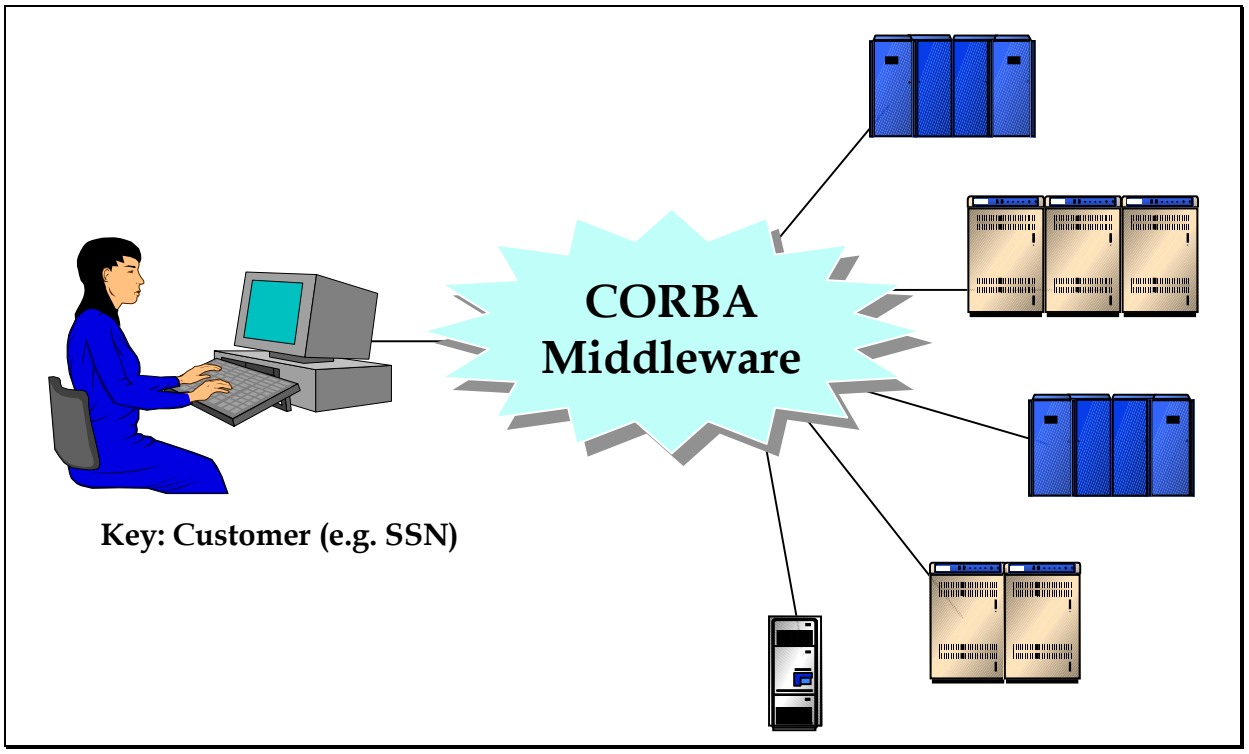


Figure 2 - Information Access After

[3] The authors are aware of no other CORBA-based application system in production use which rivals the size and scope of the ORB-based applications at Wells Fargo. However, our experience has also been that many organizations working with this technology have avoided public disclosure of their efforts because they view the technology as a competitive advantage. If any effort of similar scale exists, its secrecy has been well guarded.

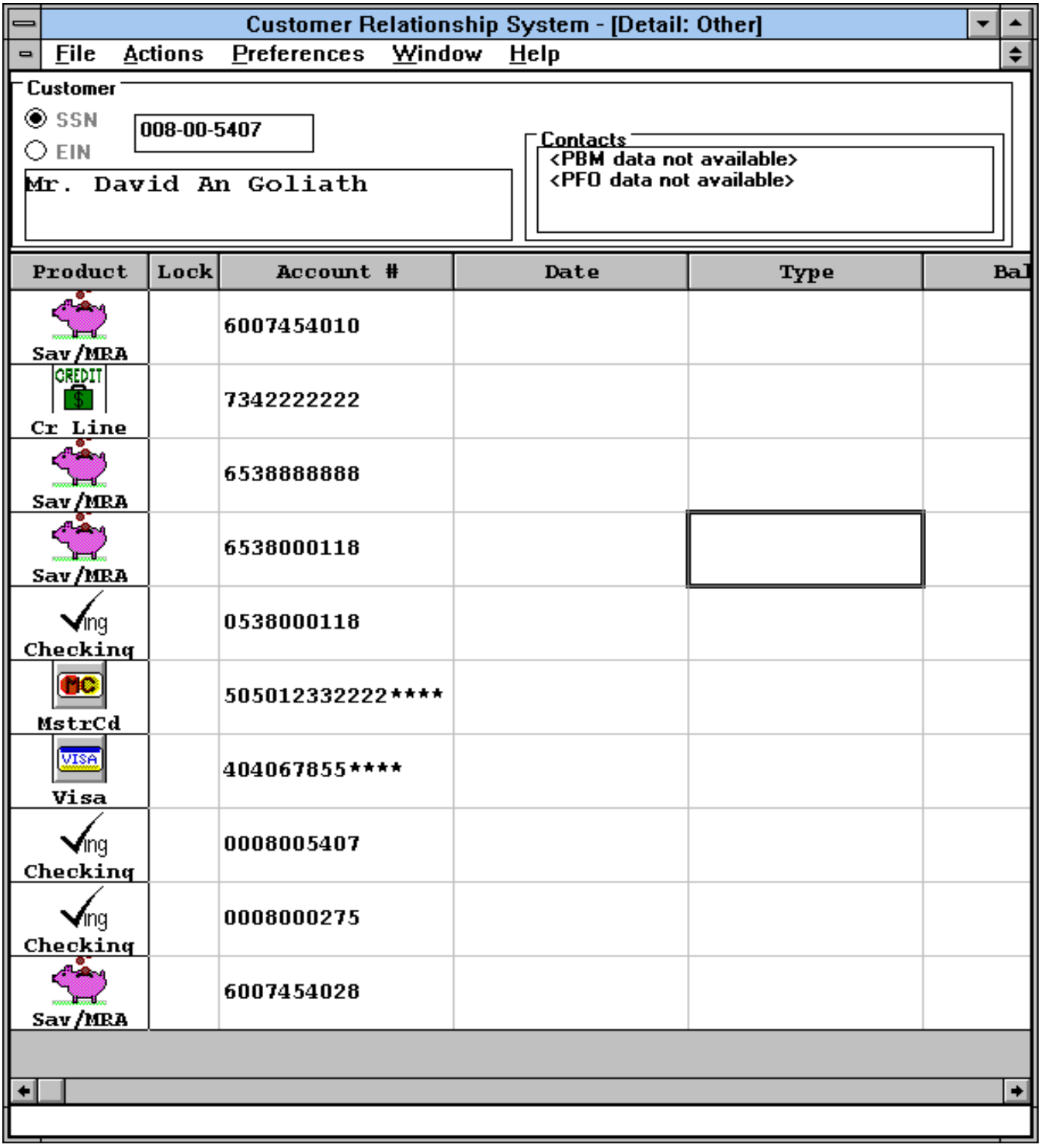


Figure 3 - The end-user's view of the CRS Application

The project team used *Object Request Brokering (*ORB) technology from Digital to build an application that allowed the service agent to retrieve a comprehensive profile of customer account information simply by entering the customer's social security number (or EIN for business accounts). The system then interrogated several systems of record including IBM MVS/CICS hosted applications, an application running on a DEC VAX/VMS system, and a Unix-based system. An organized, coordinated view of the

customer's *relationship* with the bank was returned to the user and presented through a Microsoft Windows™ GUI. This included a list of *the* accounts the customer had with the bank, their current balance, status, etc. Graphical user interface elements such as color and icons were used to organize the display and highlight certain conditions, such as overdrawn or frozen accounts. This application would be named the *Customer Relationship System*, or CRS, reflecting its function of presenting all accounts in a customer's *relationship* with the bank.

In the first phase of this project, it was not practical to provide a graphical user interface to all of the possible functions a user might want to perform on the accounts themselves after identifying them. Those features would come with time, but both schedule and funding constraints dictated a retrieval-only scenario for the first version. Eric Castain devised a very powerful extension, however. When the user wanted to actually perform transactions on an account, they simply double-clicked on that account on the screen. CRS then activated a terminal emulator window connected to the appropriate system of record. The terminal emulator was then "driven" by CRS to navigate to the appropriate screen in the mainframe application, enter the account number, and "bring up" the account in question in the mainframe application. The user then took over, using the mainframe application he or she was already familiar with. CRS took the legwork out of the process, and put the user in the right screen for what they wanted to do, which was a major improvement. Later projects would move more and more functionality into the GUI, alleviating the need for the end user to "click into" the mainframe application's native character-cell user interface. Given the complexity and difficulty of migrating away from legacy mainframe systems, this approach offered a reasonable alternative to the "Big Bang" tactic of replacing legacy systems. In fact, it is often possible to extend the useful life of existing systems using this technique, assuming that there are no other compelling reasons for retiring the legacy application.

A team was formed to build the CRS application. The team was led by Eric Castain of Wells Fargo, who has since become the bank's principal champion of ORB technology. Three Cushing Group consultants brought ORB and distributed systems experience to the team. Digital provided the ORB product itself as well as one consultant who was an expert in PC GUI design & development. Several Wells Fargo employees provided knowledge of the existing applications, and participated in the overall development. All began in early October of 1993, with the goal of implementing CRS in the very aggressive timeframe of 90 days. Of course, it would be impossible to re-engineer the systems of record in that time, but it was felt that the Digital ORB technology was powerful enough to *encapsulate* and integrate the existing systems into a working application in that time.

Designing the CRS Application Using Distributed Object Technology

The CRS application was designed from the end-user's point of view using rapid prototyping of the graphical user interface components of the application as the *drawing board*. Since this application was intended for use by Customer Service Agents, we enlisted two of them to participate in the design sessions. There were also a number of other participants from the business side of the organization involved from day one. The ratio of business participants to IS participants was close to 2:1 throughout the effort and this factor contributed heavily to the overall success of the project.

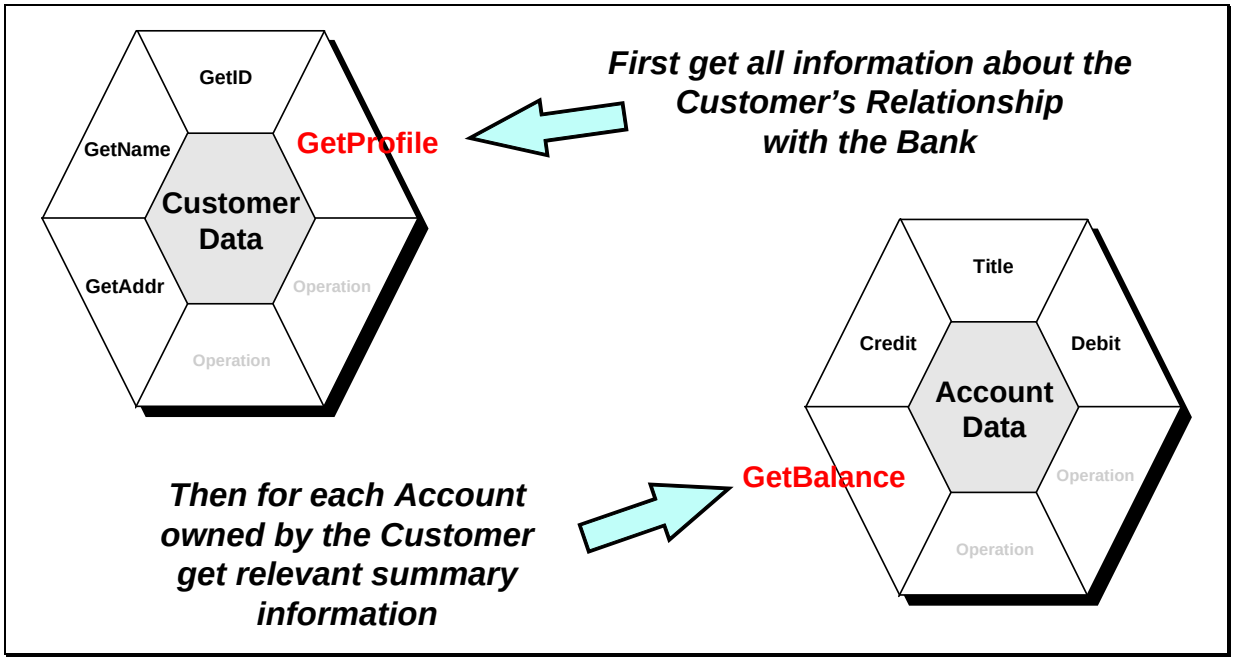


Figure 4 - Simplified View of Customer and Account Business Objects

While the GUI designer/programmer from Digital worked with the users, the other consultants from The Cushing Group concentrated on defining the Business Object model, mapping that model to the existing legacy applications' 3270 transactions, and developing the UNIX-based methods that would implement the business objects. This initial project relied heavily on *screen scraping*[4] technology to gain access to the functions and the data in the existing environment. As the object model took shape, each operation was mapped to the corresponding mainframe application screen, and data fields that would have to be accessed in order to perform its function were identified. Once this information was specified, the required screen-scraping routines were implemented by a consultant from OpenConnect Systems of Dallas, TX. The screen-scraping routines were incorporated into the ORB server program by a Cushing Group consultant.

The resulting application provided Wells Fargo's telephone customer service agents with a Microsoft Windows™-based graphical user interface to customer relationship information, as depicted above. The only necessary input from the end user is the customer's social security number.

Integrating Mainframe applications using an ORB

In 1993 when CRS was developed, no commercial object request broker product was available for IBM mainframes.[5] In our consulting experience, we have helped several large organizations to evaluate CORBA-based middleware, and have almost universally found these customers to be very concerned about an ORB vendor's support for mainframe platforms. Although at first glance, it would seem obvious that an ORB must support the platform where most legacy applications reside, we believe much of this concern to be misplaced. Out of necessity, we developed a technique for integrating mainframe-based applications into a CORBA-based system, despite the absence of the ORB itself on the mainframe. In retrospect, we

[4] Screen scraping was chosen in the interest of expedience, and at the cost of some architectural robustness and reliability. The screen scraping software was later replaced with software which communicated with the mainframe using a messaging based protocol developed internally at the bank.
[5] Digital ported it's ObjectBroker™ product to IBM's MVS operating system, but the mainframe version was not available until after the completion of the initial release of CRS.

feel this may be the best overall approach, despite the fact that CORBA is now available on the mainframe platform. More on that later – but first, let's examine the approach that was taken for CRS.

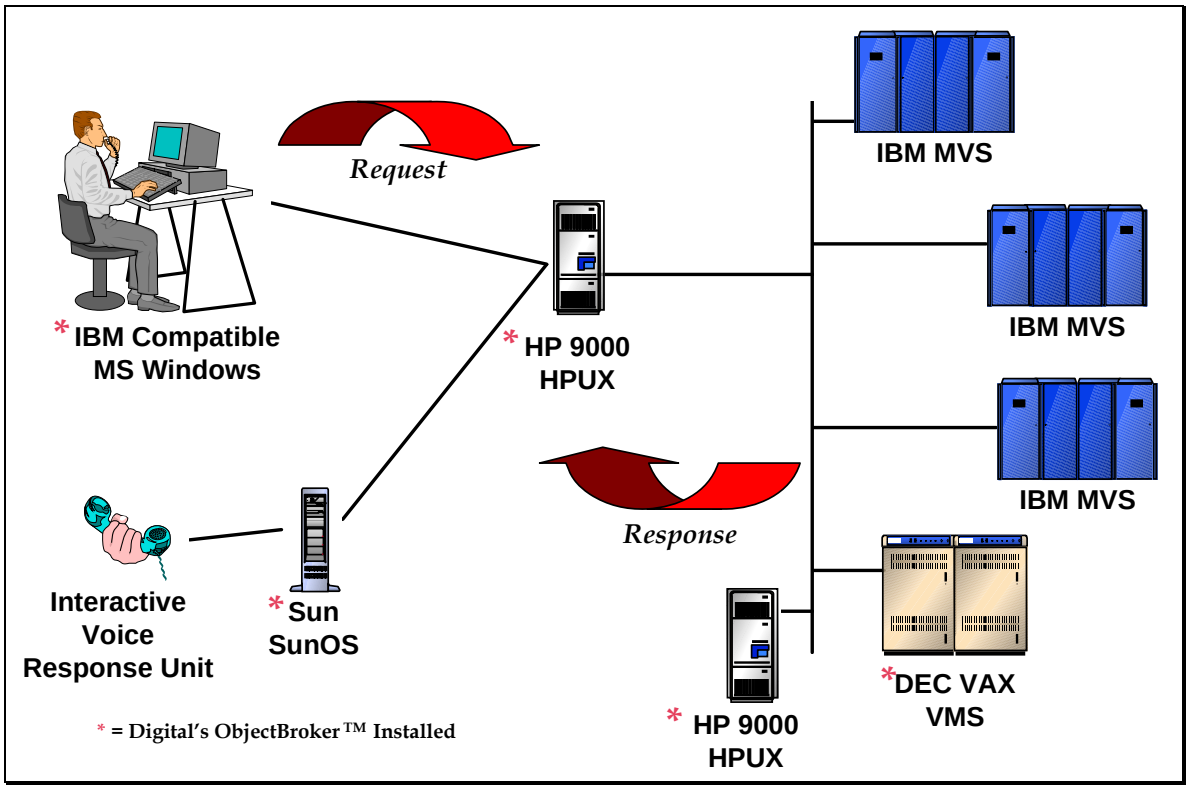


Figure 5 - Simplified System Diagram of the CRS application[6].

As depicted in Figure 5, Digital's ObjectBroker™ was used to provide communication between applications on all platforms *except* the IBM Mainframe, which ran MVS. Although the *Account* and *Customer* business objects were actually implemented by application *systems of record* primarily hosted on the mainframe platform, the CORBA object implementations for *Account* and *Customer* were placed on the HP 9000 platform running HP/UX (Hewlett Packard's Unix operating system). The CORBA servers running on the Unix platform implemented these business objects by communicating with one or more mainframe applications using non-CORBA mechanisms.[7] There are several indirect effects of this:

- The semantics of communication between the Microsoft Windows™ client and the HP/UX object server are based on a CORBA interface definition. This interface was architected to provide the client with a logical, consistent interface to customer information, independent of how those data are stored and by which applications they are stored.

[6] The first version of the CRS application offered client access through a Microsoft Windows™ GUI only. The interactive voice response unit was later added as an additional client access channel, and is discussed later in this white paper.

[7] In the first version of CRS, Screen scraping was used. Screen scraping is a technique for transforming an application's human user interface into a programmatic interface (API) by capturing data from a 3270 terminal session. Although screen scraping is a very effective rapid application development technique, it is sub-optimal in terms of reliability, maintainability, and performance. A later version of CRS replaced screen-scraping with more direct and reliable communication protocol.

- The back-end IBM Mainframe applications are not forced to change their semantics to conform to the new object interface definition. Rather, they simply expose the semantics of their original design.
- Inconsistencies and disparity between the back-end applications are hidden by the middle-tier server (the CORBA object implementation) running on the HP/UX platform. In fact, a primary purpose of this layer is to translate and hide complexities of the back-end systems and expose a single, logical, consistent interface to client applications. Thus, if the back-end application topology changes, the middle tier server must be modified, but client software need not be changed.
- The mainframe programming environment remains unchanged. The paradigm shift to distributed object technology is made on the Unix platform[8]. To the extent that mainframe programmers must expose their applications' functions for remote access, this is done with conventional and well-known mechanisms supported by the MVS platform.
- For a business object like *Customer* whose functionality is most likely implemented by several different *systems of record*, as was the case at Wells Fargo, there are really two distinct sets of *business logic*: the logic contained in the systems of record themselves, and the business rules which dictate how the functions of various, disparate systems of record must be combined into an overall solution. In this example, the latter type of business logic (the integration rules) have been separated from the systems of record themselves, and placed on a strategic platform (in this case, HP Unix).
- The back-end systems are completely hidden from the semantics of the client's communication with the CORBA server. The systems of record could, in theory, be completely re-architected and perhaps re-implemented on a different platform without modification or recompilation of the client applications.

As noted earlier, we believe that this *three-tier* approach which does not use CORBA to communicate with the back-end IBM Mainframe-based data tier has significant merit, and should be considered today, even when the chosen CORBA middleware product is available for the mainframe. Reasons include:

- The programming paradigm on the mainframe is not changed. In general, we have found that the required paradigm shift to embrace object-orientation and distributed processing is significant, not to mention costly. This approach allows the mainframe applications to operate in a conventional environment, and localizes the CORBA server development to a middle-tier platform such as Unix. If the organization's long-term goal is to embrace Unix or Windows NT as their strategic server platform, it only makes sense that the organizational development cost of adopting CORBA should be made on a strategic platform.
- The *business integration logic* (The business rules which dictate how functions of several disparate systems of record are combined to implement a single logical business object, such as *Customer*) are separated formally from the systems of record themselves, and hosted on a strategic platform. They can be easily changed as new systems are developed (or more likely, as off-the-shelf applications are purchased and integrated).
- Cost of redundancy is lower. By placing these middle-tier servers on relatively inexpensive Unix servers, they can be redundantly deployed, and fail-over features of the ORB can be used to ensure that at least *limited* functionality of the business objects is available at all times, even when one of the numerous mainframe-based systems of record is down. This is a benefit of the three-tier architecture and could be attained using a CORBA-enabled mainframe as the middle tier, but the cost would be much higher.

[8] Unix is mentioned here because it was used by Wells Fargo. Any other strategic server platform, such as Windows NT, could be used just as easily.

- Risk is reduced. Although CORBA is now available on the mainframe, support is quite limited.[9] In the authors' view, it is quite plausible that the CORBA vendors attention will be focused on newer platforms, and their responsiveness to customers who experience problems on the mainframe platform may be impaired. In contrast, mechanisms such as APPC and LU6.2 are well supported in the Mainframe marketplace, and little concern exists as to whether they will continue to be supported.

The primary counter-argument from proponents of mainframe-hosted ORB servers is that the approach we took appears to require an extra "hop" in the network A client's request must be processed by both the mainframe and the Unix server before a response can be delivered to the client. Of course, this is entirely true. However, the common assumption that a performance penalty will necessarily result has proven not to be valid so far in our experience. Our analyses have shown that the overall systemic performance of a configuration that includes this "extra physical node" is usually limited by the capability of the back-end "legacy" applications. The modern Unix or Windows NT servers running on multi-processing platforms easily keep up with the mainframe-based back-end systems in a messaging environment, even as they reach maximum capacity. We have yet to encounter a situation wherein overall systemic performance is degraded as a result of this approach.

The IBM Mainframe will remain an important platform for many years to come, and availability of native ORB technology on that platform will, of course be important. For many applications, direct implementation of CORBA objects on the mainframe platform will be appropriate and desirable. Nothing above is meant to detract from this; rather, we mean to point out that direct implementation on the mainframe platform is not the only option, and that the alternative *indirect wrappering* approach discussed above may, in some cases, actually be preferable. As a general guideline, if the objective is to eventually move the application off of the mainframe platform, indirect wrappering is likely preferable. For applications which are intended to remain on the mainframe indefinitely, a direct CORBA-on-the-mainframe approach is more likely to be appropriate.

The CRS Experience in Retrospect

The CRS application was placed in pilot in late December of 1993 and in full production use in January 1994, one hundred four days after we first arrived in San Francisco in early October. Our original (and rather aggressive) target was 90 days to full production deployment. We feel that several last-minute additions of functionality (not to mention the winter holidays) justified the slip to 104 days. Wells Fargo invested in the product licensing of Digital's ORB technology and 47 person-weeks of consulting services from The Cushing Group and Digital, and the personnel cost of approximately 3 full time Wells Fargo employee participants. The result was a working solution that significantly re-shaped the way users interacted with the bank's core systems of record.

The end users of this application were located in Concord, CA, some 25 miles east of the downtown San Francisco offices where the application was developed. The GUI interface and simplicity of the application induced a group of end users to travel to San Francisco for the purpose of complementing the developers, taking them to lunch, and presenting them with an award. One bank executive was quoted as saying that although this was not the first time end-users have "gone looking for the programmers", it was probably the first time the purpose of such a visit was a positive and complementary one!

The successful introduction of CRS notwithstanding, one might observe that no really compelling reason for selecting *object* technology over any other distributed computing approach has been presented. And indeed, the CRS application could have been developed using message-oriented middleware, remote

[9] At the time of this writing, the only ORB available on MVS and OS/400 was Digital's ObjectBroker™. The installed base for that product is quite small. Other ORB vendors such as Iona Technologies have announced support for MVS but have only just begun to deliver versions of their ORB for that platform.

procedure calls, or any other similar approach. Such is the case with the first application that anyone develops with distributed object technology.

The difference between distributed object technology and other approaches centers on the re-usability of the business object services. Since no such services exist before the first application is built, little tangible value is perceived in this area until a second application can be developed. Provided that the first application is well architected, the cost to develop other applications should be reduced by re-use of components which were built as part of the earlier applications. That is exactly what occurred at Wells Fargo[10], and probably the principal reason that Wells Fargo has been so successful with this technology.

It is noteworthy that success in this area is directly related to the quality of the object model. In the interest of meeting the very aggressive schedule objectives of the CRS project, we intentionally allowed the object model to be less than perfectly object-oriented, and we did not test the object model against *use cases* outside the immediate functional domain of CRS. The object model that supported CRS was eventually re-architected to accommodate needs of other client applications. At the same time, Digital's ORB product became fully CORBA-compliant, which also impacted the object model definition. Through this process, the development team at Wells Fargo learned to treat the object model and the software which implements it as a *critical success factor*. The investment of time and energy in developing an object model which truly represents the business process will easily be justified by the resulting re-use benefits as additional client applications are developed.

Expanding on CRS: IVRU, Brokerage, ATM, and Quotron

Fueled by the success of CRS, new ideas and opportunities emerged to use Digital's ORB technology to deliver new innovations to Wells Fargo's customers. These included:

- An *Interactive Voice Response Unit*,[11] or IVRU was added as a client for a new application, allowing Wells Fargo customers to interact more directly with the ObjectBroker™-based infrastructure
- Access to customer brokerage accounts (not part of the original CRS system) was added, which involved integrating the Tandem platform into the system
- Automated Teller Machines (ATMs) via a connection through the mainframes along with the mainframe-based *Prodigy* information service were added as clients to provide access to customers' brokerage account balance information
- A stock market data application was purchased from Quotron. This application, hosted on an IBM RS/6000 platform running AIX (IBM's Unix) received real-time feeds of stock market data from the exchanges via a network provided by Quotron. The application was encapsulated and exposed as an ObjectBroker™ server, making stock quotes and other data available to any application in the Wells Fargo network through a simple ObjectBroker™ client request. This enabled delivery of stock quotes to customers through several channels

Through these projects, several new innovations were delivered to Wells Fargo customers. But perhaps more importantly, the object model was refined, yielding a robust library of truly re-usable business object services which encapsulated the bank's core systems of record. Wells Fargo was now poised to fully

[10] The re-use of these object services and the business benefit derived from such re-use is described later.
[11] An IVRU is the generic name for a device which provides automated service to customers calling in over telephone voice lines, using a user interface based on touch-tone dialing equipment. For instance, when an automated answering system invites you to "Press 1 for customer service, 2 to place an order, or 3 to request product information", an IVRU is the device responding to your inputs.

exploit the principal benefit of distributed object technology: Re-use of business object services to satisfy a new business need at a fraction of the cost and time-to-market which would be required using a traditional, ground-up approach to solving the same problem.

The World's First Internet-Enabled Bank

When we developed the CRS application in 1993, the World-Wide-Web existed but was essentially still a tool used by research and development companies. By March of 1995, the Web (and the Internet on whole) had taken on an entirely different meaning. Wells Fargo was becoming a technology-leading bank. Management at Wells Fargo funded an effort to offer electronic banking to its customers through a Web server on the open Internet. Wells Fargo wanted to deliver on-line electronic banking services to customers over the Web, rather than limiting the site to containing product literature as was common at the time.

Sixty days later, Wells Fargo was online with real-time access to account balances via the Web, and in so doing, became the first truly Internet-accessible bank.

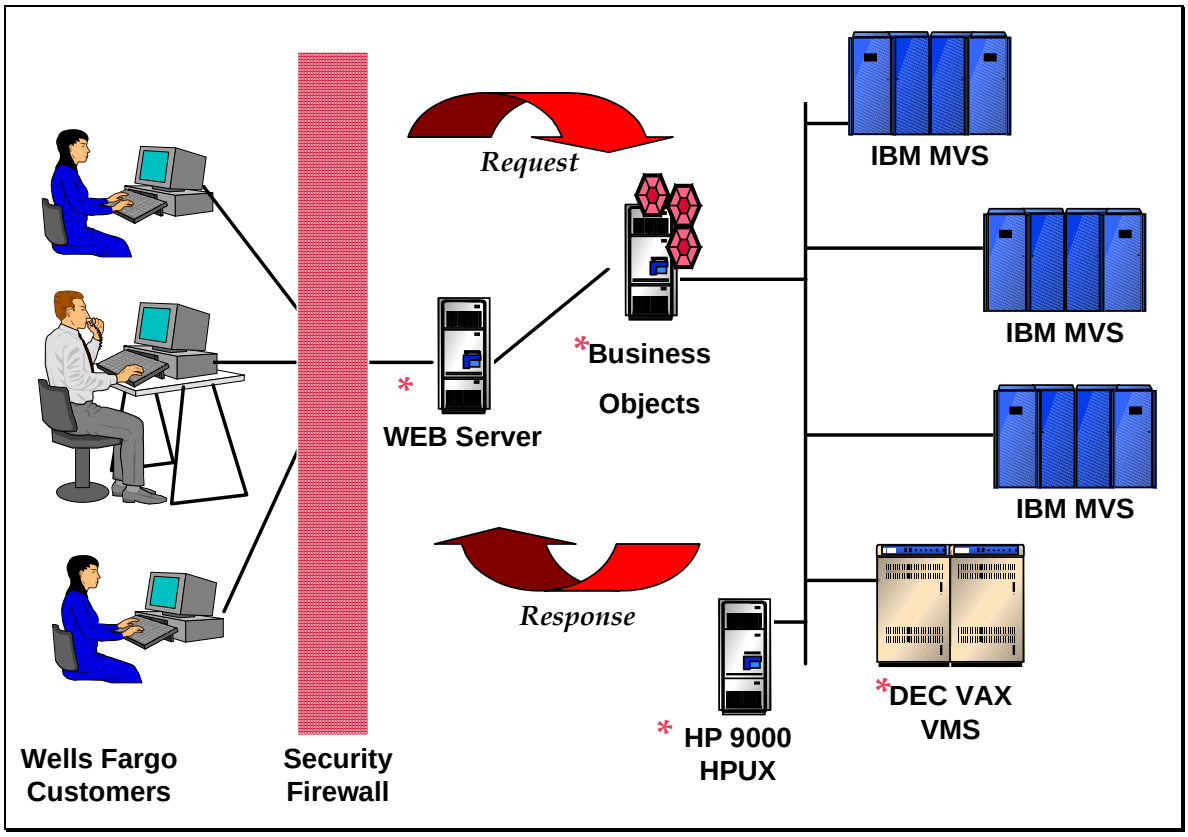


Figure 6 - Wells Fargo's first Internet Banking Solution - Implemented in 60 days.

The amazingly short 60-day project cycle was directly enabled by the re-usable business object services developed for CRS and other applications. The legacy systems which needed to be integrated had already been exposed through distributed object interfaces. To a large extent, the functionality needed was already available on the network, and if not for security issues, the project would have involved little more than hooking up a Web server as an ObjectBroker™ client, and using a CGI script in the Web page to escape to

C code to make the ObjectBroker™ client request. Of course, security *is* of paramount importance to a bank, so the customer authentication and authorization checks, firewall protection, etc. became a significant task - probably the lion's share of the project on whole.

The goal of the 60-day effort was to deliver online access to account information. Following the same incremental development philosophy which has brought success to other CORBA-based projects, Wells Fargo later went on to deliver numerous other banking functions to customers through the Web interface. These newer, more expansive Internet banking capabilities are also built entirely on top of a CORBA-based internal software integration architecture. Wells Fargo's Web site can be found at http://www.wellsfargo.com. The pages which actually access account data through the CORBA infrastructure are (of course) available only to Wells Fargo customers. IT professionals evaluating use of CORBA in their own organizations have actually opened Wells Fargo checking accounts solely for the purpose of being able to use this unique Web site!

It is interesting to compare Wells Fargo's Web solution to other banks' competing solutions. One of Wells Fargo's competitors recently (mid 1996) announced availability of their home banking product, which includes a bill payment service through its Web site. Given that the competitor's solution was developed more than a year later than Wells Fargo's, one might assume that the competing site would have the benefit of more mature development tools and environments, and therefore be generally more technically advanced. Yet the opposite appears to be the case. Computerworld's June, 1996 issue reported that the competing bank's Web site, through CGI callout's, sends e-mail to bank employees who, apparently, later manually key transactions into the mainframe-hosted application system. In contrast, Wells Fargo's solution is fully integrated and automated, and provides real-time connectivity to the actual systems of record, without need (or cost of) human intervention. We regard this as evidence of the viability of distributed object technology for complex, disparate systems integration. The robustness of their Web solution serves to prove that Wells Fargo has achieved its goal of being the technology-leading bank.

Making Object Re-Use Work in a Large Enterprise

Re-usable software components have no inherent benefit over any other kind of software unless the organization which owns them is successful in re-using them. While that may seem obvious and self-explanatory, real experience has proven that developing a library of re-usable components -- *and then actually realizing a dramatic reduction in development cost and time to market as a direct result of such re-use* – is more difficult than one might guess. Traditional management structure does not create an incentive for people to re-use existing components. Hence, they often create new, redundant components, thereby adding to the disparity between systems. The monolithic, redundant, dysfunctional legacy applications are replaced with redundant, dysfunctional distributed systems, each comprised of components that are theoretically re-usable except that they are never actually re-used. Unfortunately, such has been the case with several organizations that have adopted CORBA.

Over the last several years, The Cushing Group has worked with several large organizations which have adopted distributed object technology. We have yet to come across any organization which has been as successful (in terms of effectively exploiting re-use opportunity for CORBA-based business object services) as the Wells Fargo team. In the view of the authors, much of Wells Fargo's success is not attributable to any technology or software function. Rather, it is owed to Wells Fargo's willingness to modify and adapt its managerial approach to actively promote the re-use of software components between different groups of developers, which is inherently contrary to the culture which exists within many large companies.

The key to enabling re-use lies in having, maintaining, and sharing a well-architected Enterprise object model. The authors believe that Wells Fargo's success is owed largely to the fact that they invested in developing such a model, and are very careful to maintain, extend, and use it as effectively as possible. The

remainder of this section will describe some techniques they have employed to keep the model consistent and to ensure it is used effectively. The model itself is discussed later.

As with any other distributed computing middleware, a programmer can "hack out" a CORBA Interface Definition Language (IDL) file[12] based on his or her knowledge of what the application must do. This might be done in the matter of a few minutes. The programmer might then go on to build client and server applications using that IDL. ***This is exactly what must be avoided***. The IDL defines the interface through which an object is used. Or re-used. An object interface whose design is based solely on contemplation of the requirements immediately at hand may fail to make information or functionality available which another, future application would need. For example, when defining the interface to a *BankAccount* object type, one must consider not only what functions and data the client program which plans to use that object may require, but also what other functions and data future applications may require.

Almost invariably, or so it seems, if left to their own devices, programmers will write their own IDL to define a given object rather than use one that already exists. So a need exists to both ensure that objects are well designed in the first place, and that they are used effectively thereafter. Furthermore, when a function which does not exist is needed, care must be taken to ensure that the function be added or fit into the most appropriate existing object definition before simply creating a new, perhaps overlapping or redundant object type. Simply put, Wells Fargo's success is owed principally to their realization that these things don't happen by themselves, and that an organization must be proactive to make them happen.

Wells Fargo has commissioned an *ORB Coordination Group*. Essentially, the function of this small group of people is to be the keepers of the Enterprise Object Model. Rather than allowing anyone so inclined to simply define, build, and use object interfaces, all development groups using the object middleware are required to work with the ORB Coordinator to ensure that their object definitions are not incompatible, overlapping, or redundant with those another group or developer may be working with. The ORB Coordinator is *not* a dictator who stipulates the content of the object model. (The dictator/chief architect approach has been tried in other companies but never scales because one person cannot have sufficient subject matter expertise to know what the overall object model for a large organization must contain). Instead, the ORB Coordinator's role is to be the *librarian* who knows what object interfaces different groups are working with, and to facilitate sharing between them. So, for instance, in response to a request to add a new *NoLoadFund* object to inherit directly from the *Account* object in the shared Enterprise object model, the ORB Coordinator might respond by encouraging the requester to instead work with the group which is already building a *MutualFund* object, and extend its semantics to cover no-load funds. Another aspect of the job is to identify areas where different developers or groups seem to have inconsistent views as to how the object model should be used, and to facilitate reconciliation of those inconsistencies.

This coordination function may sound obvious, simple, and trivial. The truth is that it is of paramount importance, and most organizations we've dealt with don't make the investment. Something is to be said for the success Wells Fargo has seen, and we believe much of it to be attributable to this one factor alone.

The Enterprise-Wide Object Model

An object model defines the re-usable software components (business objects) in a distributed object system. The object model specifies what the objects are, and stipulates the semantics they use to communicate with other software. So in a banking system, for instance, examples of objects are *CheckingAccount* and *RetailCustomer*. For each object, the model defines its relationship to other objects So, for instance, it is probable that a *RetailCustomer* object is capable of owning one or more *CheckingAccount* objects. The model defines what data associated with the object are made available to other applications – a *CheckingAccount* object might have a current balance, an overdraft limit, and so

[12] The IDL file defines the semantics of communication between client and server, in a syntax standardized by the CORBA specification.

forth. The model also defines the *functions* which the object can perform. A *CheckingAccount* object might support *Deposit, Withdraw,* and *Transfer* functions, while a *RetailCustomer* object might instead support functions such as *ChangeAddress.*

The model must be shared among the group of developers who seek to integrate software. In other words, if group A has one object model, and group B has a similar (but different) object model which addresses some of the same functions, their software will be incompatible. If the long-term objective is to achieve seamless, enterprise-wide integration of systems, enterprise-wide coordination of the modeling effort is necessary. This is not to say that there must be a single model developed by one person. Quite to the contrary, it is desirable to have different organizations use their subject matter expertise to develop object models within their functional domain. When two different groups overlap in their effort and define slightly different semantics *for the same object types,* problems arise. The most successful approach is for several different groups or individuals to *contribute* to the development of a coordinated, enterprise-wide object model.

For an organization to be successful in using distributed object technology, it is *essential* that the model be well designed, consistent, and that it be based on proper object-oriented design methodology. The single largest factor (in the authors' view) to the failure of some companies' efforts to use CORBA as a systems integration technology is that they fail to invest in (or realize the need for) a proper model. Many companies who think they are developing distributed object systems are actually using CORBA-based middleware as a glorified remote procedure call mechanism. The resulting applications will likely provide useful point solutions, but will not integrate with one another to deliver a unified solution.

Wells Fargo has learned the importance of this from experience. The original object model used to build the CRS application met the immediate needs of CRS, but failed to consider other requirements. Wells Fargo has evolved its object model to eliminate these deficiencies, but at some cost. We interviewed Mr. Castain for this white paper, and solicited his advice to other organizations. He suggested that others planning to develop object models for distributed systems view the process in three parts:

- Modeling (defining interactions between objects) in a distributed environment. Mr. Castain noted that commonly available object design and analysis texts fail to consider design issues unique to *distributed* object technology.
- Understanding the business process (both existing and desired) sufficiently well that an accurate object model can be derived
- Combining the above so as to specify a business object model which represents the business process in an accurate and extensible way, and fully exploits available distributed object technologies such as CORBA.

Mr. Castain was emphatic in saying that the only way to achieve all three items is through experience, which can be gained only by willingness to develop a bona fide production system with the knowledge that portions of the environment will require significant redesign as the organization learns the trade-off's and interactions which must be understood to succeed with the aforementioned three steps. (Mr. Castain specifically commented that pilot projects, demos, and proof-of-concept projects don't adequately expose the interactions which must be understood.)

Mr. Castain also noted that the model is constantly evolving. The success of CRS led to several follow-on projects which added functionality and incorporated additional business processes. In support of this, the object modeling effort requires about 2.5 full-time senior team members. These people support development teams working on several "product channel" client and server development efforts. It is recognized that the object modeling process is an essential component of the ongoing application development, and therefore, enhancements to the object model are funded as required. Despite the fact that Wells Fargo has now become completely self-sufficient with several in-house experts on distributed object technology, Wells Fargo periodically asks senior-level Cushing Group consultants to review extensions and enhancements made to the object model.

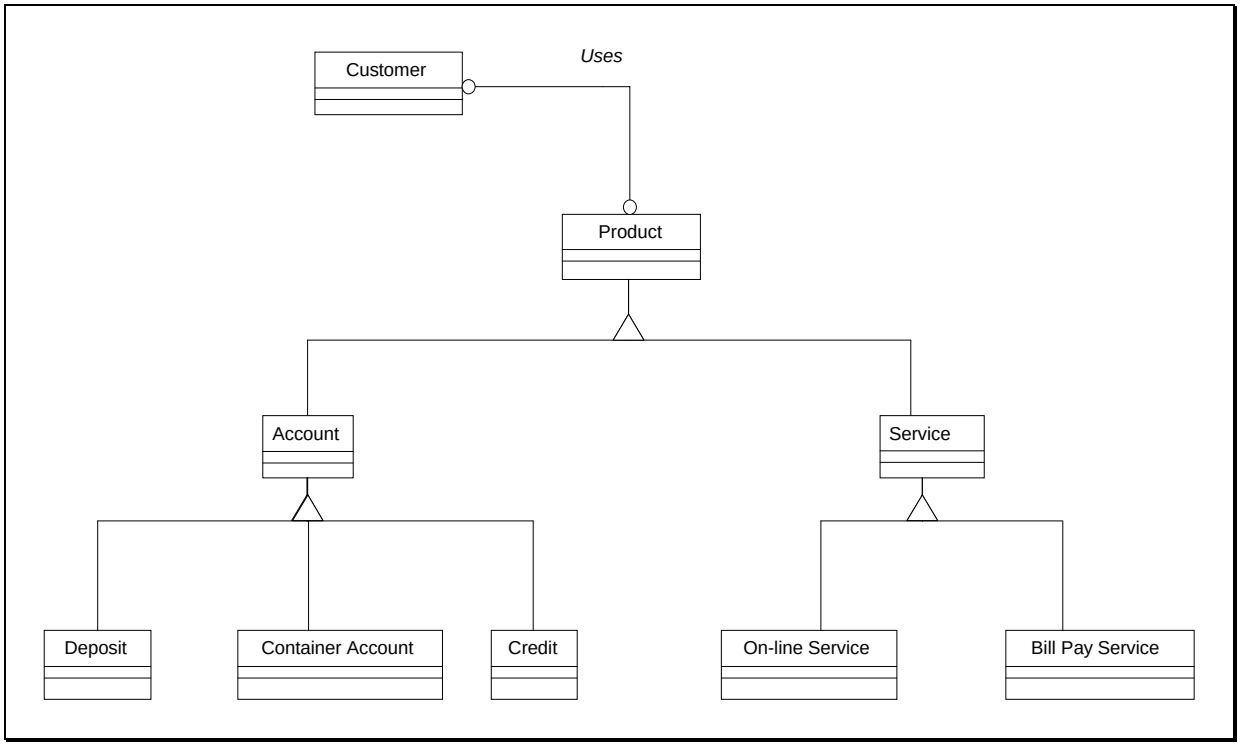


Figure 7 - High Level Model for banking Business Objects

Wells Fargo considers the detailed content of its banking object model to be highly confidential and proprietary. After all, it is the model which directly enables the integration of systems that gives Wells Fargo a significant advantage over their competitors. Therefore, we are not at liberty to share its detailed contents in this white paper. After some negotiation, we obtained permission to print the object diagram in Figure 7, which shows a simple example of what an object model looks like. Some of the details have been eliminated in deference to Wells Fargo's request for confidentiality.

The diagram in Figure 7 shows how a bank's products relate to one another. Products consist of accounts which can be owned by a customer, and services, such as bill payment, which can be used by a customer. The actual object model used by Wells Fargo contains much more detail than is included here. For each object type, the set of possible functions (or *Operations*, in CORBA parlance) which can be performed on the object are defined. Each of these operation definitions includes a set of input and output arguments which must be specified by the client of the object, similar to the definition of a callable function in a procedural programming language. Also defined in the model is the definition of *attributes*, or data values stored to describe an object. For instance, a *CheckingAccount* object type might include attributes such as *LedgerBalance* and *OverdraftLimit*. The model further describes relationships between object types, as is suggested by the depicted relationship between the *Customer* and *Product* object types in the diagram. Again, the diagram in Figure 7 is rather simplistic, but gives one the general idea of what an object model consists of.

Keeping Track of the Model

The Enterprise Object Model becomes an asset of inestimable value. Creating, maintaining, and documenting the model is of paramount importance. In the early days at Wells Fargo, we designed the model on flip-chart pads during an off-site "architecture summit". It's amazing that we managed to keep it

straight with such a primitive technique, and it would clearly be impossible to do so now given the ever-increasing size and complexity of the object model, and its importance as a corporate asset.

Wells Fargo has adopted Rational Corporation's *Rational Rose* object modeling tool to maintain the object model. Rose makes it possible to keep the model, which is now quite large, documented and easily maintained. Rose allows one to design an object model graphically, and CORBA IDL can then be generated. Rumor has it that Wells Fargo has also begun to work with Rational to identify opportunities to further automate the software development process, assuming a model contained in Rose is the starting point for all development.

Managing the Run-Time Environment

Wells Fargo now depends on its CORBA-based infrastructure as a key component of its operational systems environment. Like other mission-critical systems, the ORB servers support the bank's ability to conduct business. Service outages are simply not acceptable. To support such mission-critical use, the run-time environment must be managed and controlled. Automated mechanisms must be in place to monitor the status of ORB servers running on several different computers. If a server process fails, an automated mechanism must respond by re-starting the server, perhaps on a different network node. The failed server's clients must be redirected to an alternative server, and in many cases, real-time human intervention needs to be initiated (the system needs to be able to alert someone about problems that can not be automatically resolved). Client demand for servers must be load-balanced across several network nodes, and their usage must be tracked. Instrumentation must be maintained on the overall system, to identify bottlenecks and allow the overall run-time environment to be tuned for optimal efficiency.

Evaluating the commercial ORBs (e.g. CORBA products) in terms of their support for requirements such as these is easy: none of them have any. And unfortunately, all of the leading ORBs are, out of the box, unsuitable to support a high-volume, production environment such as that which Wells Fargo has constructed.

Out of necessity, Wells Fargo has designed its own *ORB System Management Facility*, or OSM. OSM is a complete management tool which augments Digital's ObjectBroker™ to provide run-time instrumentation, dynamic run-time management of server processes, load balancing management, exception handling (run-time errors can be automatically escalated for human intervention through an alarm system), and complete reporting of performance metrics to facilitate tuning. OSM provides all of these functions through what is essentially an internally-maintained add-on product. OSM functions today with Digital's ObjectBroker™, but was designed to be adaptable to other vendors' ORB products.

It is interesting to note that OSM itself is a distributed application based entirely on Digital's CORBA implementation. Not only does OSM provide a mechanism to manage distributed object applications in a production environment, but it is itself a production distributed object system, and OSM can be used to manage the OSM environment on the network as well as to manage business applications.

Wells Fargo developed OSM to meet a critical need, and continues to maintain OSM for its own use. However, Wells Fargo is in the banking business, not the software business. At the time of this writing, Wells Fargo and The Cushing Group were discussing the possibility of offering OSM as a commercial product for use by other organizations, as a formal product to be offered by The Cushing Group. It is clear that in order to deploy distributed object systems in large-scale production use, other organizations will require this type of management facility. At this time, it is not clear whether the ORB vendors will respond by meeting this need directly, or if an add-on product like OSM will be required in order to use CORBA in a large-scale production environment.

Organizational Issues: Half the Battle

Having been featured speakers at ObjectWorld and other similar trade events, the authors have been exposed to literally hundreds of IT Managers who are evaluating the use of distributed object technology for their organizations. Frequently, our experience has been that people conducting such evaluations focus entirely on the technology itself. The questions we hear almost always revolve around which ORB product has a better IDL compiler, or whether CORBA V2.0's Internet Inter-ORB Protocol (IIOP) should be relied upon, etc. Seldom have we encountered organizations seeking to use this technology who exhibit a concern about the effect of a fundamental paradigm shift on the organization.

CORBA is a technical specification which ORB vendors must conform to. We recommend that organizations seeking to exploit the opportunity offered by distributed object technology not initially concern themselves with CORBA. Rather, they should very carefully evaluate what it will mean to transform their organization to one which can effectively develop *component-based, distributed, object-oriented* software, and then go on to effectively develop business solutions based on the re-use of those software components. Technical specifications like CORBA are an important detail involved in the process – but are nothing more than a detail. The challenge is in transforming the organization to be able to fully exploit the technology.

Wells Fargo's success is largely attributable to their management's willingness to explore different management approaches to create an environment conducive to successful re-use of application software components (see Making Object Re-Use Work in a Large Enterprise, Page 15).

The myriad of issues that an organization should consider in making this transition could probably fill a book – never mind a section of a white paper. The following are just a few highlights:

- **Thinking in terms of components**. For the entire history of computers being used for business information systems, the software development paradigm has been one in which a business problem is identified, then a solution is built; essentially from the ground up. Advancements such as database management systems have provided useful *platforms* upon which to build business applications, but these are *technology infrastructure* platforms. The business application logic has always been built, in its entirety, to support a given problem.

  Component-based software development involves building *business solution* components - chunks of re-usable software which solve some aspect of a business problem. These components are combined in new and different ways to solve each business problem that arises. This means that the design process must begin not with "What software will solve the business problem at hand", but rather, with "How can the business problem at hand be de-composed into smaller, more generic sub-problems, and how can the requirements of solving those sub-problems be defined in a way that yields components that can be combined in one way to meet the problem at hand, yet also be later recombined – perhaps with other components from another source – to solve a different problem?". Simply put, this is counter-culture for most IS organizations. The emphasis on solving the problem at hand often drives these organizations to a *"quick & dirty"* approach which will never yield re-usable components. We have found that in many cases, it is easier to teach someone with a formal software engineering background to develop application components than it is to mentor an experienced application developer to become an application component developer.

- **Promoting re-use.** Human beings, generally speaking, are motivated by their basic need for self-esteem and by the reward system which they work with. Talented software professionals are well known for their willingness to work through the late hours in order to construct what, to them perhaps, is their own masterpiece - a complete, robust application system. Managers tend to reward those who write the most [working and supportable] code.

  In the component environment, writing more code is no longer desirable. Programmers must be rewarded not for how much software they develop, but rather, for the creativity they

exhibit in re-using existing software components to solve business problems that arise. Those who do develop components should be rewarded not only for the suitability of the component to its "intended" purpose, but also for the creativity they exhibit in designing the component to be very generic and easily adapted for other uses. Again, all of this is radically different from the culture which pervades most IS organizations.

The Internet application that was built at Wells Fargo is an example of these concepts. The goal was to quickly build the application and be the first to market. This was only attainable by the re-use of components already built for the customer service agents' desktop application.

To insure that this process continues at Wells Fargo, the ORB Coordination group reviews all operations being proposed for a particular client application team with the other teams to insure that it is appropriate for them as well. While this lengthens the specification time for an object definition, it also raises the likelihood that the object will be easily reusable by all of the teams. So far, this process has worked exceptionally well at Wells Fargo with only a small handful of operations having to be re-worked to make them more general.

Organizational Development & Readiness

Organizational readiness is a key success factor. Distributed Object Computing is more than the middleware tools which provide for communication between applications. Rather, it embodies an entirely new approach and methodology for application software design. Wells Fargo made the following investments to support its use of distributed object technology:

- **Formal training.** Since 1993, The Cushing Group has trained over 100 Wells Fargo employees on the use of CORBA and Digital's ObjectBroker™. Our hands-on ObjectBroker™ programming course is routinely offered by Wells Fargo's training department.
- **Mentoring.** Both Cushing Group consultants and experienced Wells Fargo employees have been used to formally mentor other groups within the bank on the design of ORB-based applications and how to take advantage of ObjectBroker™ features. This ensures that the technology is applied in a uniform and concerted manner.
- **Formal re-use coordination.** A key success factor at Wells Fargo has been the ORB Coordination Group. (See Making Object Re-Use Work in a Large Enterprise, page 15).

Re-Alignment of Resources for a New Approach to Computing

If an organization seeks to solve two distinctly different business problems, the obvious thing to do (in a traditional managerial structure) would be to organize two separate groups, each with its own manger. The first group would solve Problem A, and the second would solve Problem B. If the technology of choice is client/server, then it would stand to reason that Group #1 would develop clients and servers to solve Problem A. Similarly, Group #2 would develop clients and servers to solve Problem B.

One might hope that these two groups, managed by different people, would look for commonality in the problems they were each solving, and perhaps collaborate on joint development of servers (business objects in a CORBA model) which could be used by both groups. But one can *hope* all that one wants – that won't change human nature. Our experience has been that in such an organization, there is very seldom much sharing of object servers.

Wells Fargo found that a better approach is to structure the organization in terms of one or more *ORB Server Engineering* groups, whose sole purpose is to create servers. They do not write client programs, excepting perhaps simple ones to test their servers. Their *customers* are one or more groups whose charter is to develop client programs to solve a particular business need. These groups submit their requirements to the ORB Server Engineering group, whose funding may be tied directly to delivery of working servers to meet the needs of these client development groups. In this model, the manager of an ORB Server Engineering group is motivated to aggressively work to discover overlaps and commonality of functional requirements, and to develop the most generic and re-usable components possible.

This simple technique of organizational structure has worked very well for Wells Fargo, and appears to have had a marked impact on their success with ObjectBroker™.

Validating the Approach: An Outside Audit

With the use of ObjectBroker™ growing at an exponential rate in Wells Fargo, and after strong initial success, it was realized that the trend is moving rapidly toward reliance on CORBA as a standard software integration mechanism for several key applications. Although good success has been enjoyed thus far, Wells Fargo does not take lightly the importance of risk management, and it was felt that the CORBA-based approach should be thoroughly audited before allowing its use to become more widespread within the bank.

In the spring of 1996, Mr. Dudley Nigg, Executive Vice President of the Direct Distribution Group at Wells Fargo, commissioned *KPMG Peat Marwick and Hewlett Packard* to conduct an in-depth evaluation of the prudence of using CORBA as a tool for high volume transaction processing. KPMG and HP's assignment was essentially to answer two questions:

1. Was Wells Fargo prudent in choosing CORBA as the basis of its extensive systems integration activities, and is it prudent to continue doing so?
2. If the answer to Question #1 is affirmative, was the selection of Digital's ObjectBroker™ appropriate, and should Wells Fargo continue to rely on Digital's product?

KPMG conducted an in-depth analysis of the systems that have been developed, considered various risk factors, and interviewed most of the bank employees, consultants and contractors involved with the ORB-related work. The study was completed and its findings were delivered to Wells Fargo management on August 5, 1996. HP was commissioned to do an analysis of the scalability of ORB based applications. The findings were as follows:

On question #1 (use of CORBA):

> KPMG found that Wells Fargo's selection of CORBA-based object-oriented systems integration was sound. After evaluating the risks and benefits, KPMG recommended continued use of CORBA technology.

On question #2 (selection of Digital's ObjectBroker™)

> After evaluating several competing products, KPMG found that Digital's ObjectBroker™ was a proper choice for Wells Fargo. During the evaluation, KPMG and HP evaluated ObjectBroker™'s performance and scalability, and found them to be adequate and in some respects superior to some alternative choices. As with all scalability models, there are definite limits to the current implementation. Wells Fargo is working with Digital and HP to better understand these limits and engineer solutions to the constraints.

Wells Fargo Today: Banking Leadership Enabled by Technology

Wells Fargo has clearly become a technology leader in banking. We believe that distributed object technology has played a key role in helping Wells Fargo to achieve this leadership position. Although the original CRS application has now been replaced with an expanded, more versatile application, the same distributed, object-oriented architecture continues to be the mainstay of Wells Fargo's continuing efforts to deliver innovative new channels for customers to conduct business with the bank.

Large-Scale Distributed Object Technology Leadership

Since it's inception, the Wells Fargo Bank's Web-based banking service has grown to over 100,000 enrolled customers. Given this growth rate, the bank has so far not found it necessary to advertise this service beyond the Wells Fargo World-Wide Web home page, and the various reviews it gets in several publications. The enrollment figure of 100,000 customers is therefore quite impressive for a product with limited traditional marketing effort.

The OSM facility (see Managing the Run-Time Environment, page 19) can be used to track and report various instrumentation statistics. At the time of this writing, Wells Fargo's ObjectBroker™ servers were processing as many as 200,000 business object invocations per day (e.g. operations such as *GetBalance* on business objects such as *CheckingAccount*) in the production environment, and at peak periods, as many as 300 simultaneous Internet customers were logged into the Web site. The ObjectBroker™ middleware actually processes upwards of 750,000 CORBA method invocations daily. The difference between the figures results from the need for several processes to interact with one another in order to process a single business object operation. The Wells Fargo team anticipates continued aggressive growth with the need to support several million business operation invocations per day, based on projected growth estimates for 1997.

CORBA has from time to time been criticized in the industry as being "not ready for prime time" or "unable to support production-scale transaction volume", and has fallen victim to a host of other subjective, unsupported opinions. Wells Fargo appears to have disproved these arguments.

Distributed Object Technology in the Future

Wells Fargo has already become expert with CORBA and it is likely that they will continue to use their investment in CORBA-based, reusable business objects to help them offer their customers new and innovative products and services. Wells Fargo and a handful of other pioneering organizations have proven the merits of distributed object computing. It therefore comes as no surprise that the software industry on whole has now begun to embrace distributed object computing as the mechanism which will allow Internet/Intranet and "component" technologies to interface with enterprise-scale business information systems. (For a more detailed perspective on the anticipated merger of component, Internet/Intranet, and distributed object technologies, please refer to The Cushing Group's separate white paper on that subject, which is also available from The Cushing Group's web site at http://www.cushing.com)

Wells Fargo's technologists, meanwhile, continue to both identify new, innovative ways to exploit their existing investment in CORBA-based business objects, as well as to explore emerging technologies to determine how they can be used to help Wells Fargo offer the best possible products and services to its customers. We're not at liberty to disclose the details of Wells Fargo's advanced development efforts, but suffice it to say that we feel Wells Fargo is likely to maintain its leadership position in this area for quite some time.

Conclusions

Distributed object technology is an enabling tool which has helped Wells Fargo to become a technology leader in banking, and to deliver innovative, competitive electronic access channels to its customers. What has occurred at Wells Fargo is best characterized not as the adoption of CORBA, but rather as the realignment of the organization to embrace a service-based, object-oriented distributed computing architecture.

The Financial Services sector is essentially an information industry. As the general public becomes more technologically astute, the quality and robustness of financial institutions' computer systems will emerge as a principal competitive advantage factor, rather than just an operating expense as IT has been traditionally viewed. Wells Fargo's willingness to embrace the latest technologies and adjust the organization to fully exploit them will help position the company as a leader, as the financial services industry continues to evolve.

Many industries will undergo revolutionary change as the shift from an industrial economy to an information economy continues. Companies will succeed, in part, by realizing which technologies can leverage their business, then being proactive to exploit them effectively. Whether the opportunity be distributed objects, components, the Internet, or some other yet-unknown technology, they key will not be the technology itself, but rather the ability of the company's management to understand the technology and how to exploit it. Wells Fargo has proven its readiness through its experiences to date with distributed object technology.

About the Authors

Erik S. Townsend

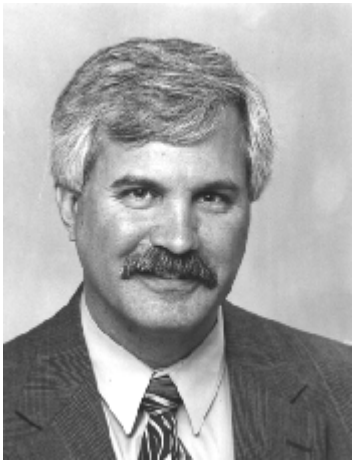
Erik Townsend is the President of The Cushing Group, Inc. His consulting career has focused on distributed Computing Middleware and Object-Oriented technologies since 1983. Mr. Townsend has served in the roles of chief architect and senior consultant in the development of complex business solutions in the finance, discrete manufacturing, and telecommunications industries. In these roles, he has successfully delivered business solutions based on distributed object computing into production for major Fortune 500 corporations.

Mr. Townsend was instrumental in introducing Distributed Object Technology to Wells Fargo in 1993, and participated during the early project phases to define the object model and help set the overall technical architecture. He continues to consult with Wells Fargo on an occasional basis, primarily with regard to emerging technology strategy.

As a consultant to Digital Equipment Corporation, Mr. Townsend was a key contributor to the architecture and design of ObjectBroker, the industry's first Object Request Broker product, first shipped in 1991.

Mr. Townsend has been widely acknowledged as an industry expert on Distributed Object Computing, and is a frequent speaker at such events as Object World and Object Expo. He has also published several articles and papers on distributed object computing and legacy systems integration.

Michael L. Ronayne



Mike Ronayne is the Vice President of Information Systems Consulting and a Principal in The Cushing Group, Inc. Mr. Ronayne specializes in executive and senior IS Management consulting in areas related to the benefits and the value of distributed application software, client/server systems, and computer information systems downsizing. Mike works across the scope of The Cushing Group's clientele, providing software architecture and design leadership. In addition to his role as Vice President of IS consulting, Mr. Ronayne is also responsible for developing the seminar and training curriculum at The Cushing Group, Inc.

Mr. Ronayne was The Cushing Group's lead consultant in the Wells Fargo project discussed in this white paper. Mike was the lead technical architect for the initial project and contributed heavily to setting the overall technical direction and defining Wells Fargo's business object model.

Prior to joining The Cushing Group Mr. Ronayne held various senior roles at Digital Equipment Corporation. As an Information Architect, he was a key member on the technical advisory team that designed Digital's next generation Information Systems Architecture. Throughout his career Mr. Ronayne has contributed to the definition of Digital products including Digital's CORBA implementation, ObjectBroker®.

