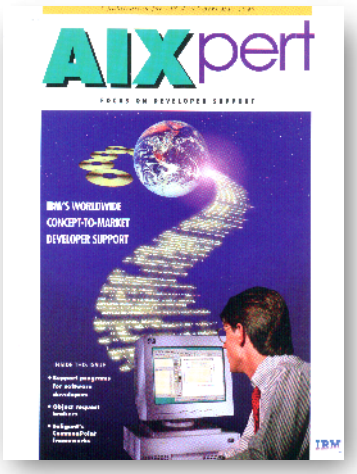
AIXpert DIGITAL VERSION IBM

TABLE OF CONTENTS



MAY
1995

AIXpert BACK TO MAIN MENU

Developer Support Supermarket

In November 1994, IBM created a new worldwide support organization for people wishing to develop applications using IBM's products and technologies. The new organization brings together top people from IBM's key areas to form a unified, "one-stop shopping" support experience for developers on IBM's platforms. Solution Developer Operations (SDO) initially provides support for Personal Software products, RISC System/6000® (including AIX®) and Power Personal systems. Other areas may be added in the future as business needs arise.

This issue focuses on the new SDO organization to give you a flavor of what it provides. It includes articles from some key areas of the organization, but certainly not all areas. We invite you to become acquainted with SDO—get connected to IBM's developer support and take advantage of IBM's know-how in software development and worldwide product marketing.

Beginning the issue is an article about the SDO home page on the World Wide Web. You'll get a little background about the Web, learn some of the information offered on the SDO home page, and find out how to get connected. Then you can learn about IBM's marketing and technical support services for solution developers—a great mix of programs—and find out how to take advantage of the program that is designed for your needs as a software developer. Find out about one such program in detail in the article about the Solution Partnership Center, and about another in the AIX Developer Connection article. You'll also find information about other programs, such as the NetView Association and the Power Personal Developer Toolbox.

But this issue is not all about the SDO organization. You will find two articles discussing aspects of CORBA-compliant Object Request Brokers (ORBs). One article discusses the features of ORBs using a specific product implementation for examples. The second shows how to develop and use before and after metaclasses with IBM's System Object Model. Continuing in the object vein, we also have two articles from Taligent®. One is the conclusion of an article begun in the February issue of *AIXpert* about building frameworks. The second discusses Taligent's CommonPoint™ application system targeted to ship in mid-1995.

Broadening the scope of this issue is an article that explains how to create a robust highly available NFS system using HACMP/6000™. And for hardware buffs, we have an article that provides configuration tips for the IBM 7318 Terminal Server. Rounding out the issue is the popular AIX Questions column.

George Noren

George Noren, IBM Corporation, Internal Zip 4103, 11400 Burnet Road, Austin, TX 78758. Internet: geo@austin.ibm.com. Since joining IBM in September 1979, Mr. Noren has written manuals for System/34, System/36™, and AIX on both the RT® and RISC System/6000® platforms, and was a member of the InfoExplorer™ design team. He has also worked as system administrator for several AIX server machines and their clients, and is currently responsible for the Prototype Evaluation Labs in Austin. Mr. Noren studied engineering at Illinois Institute of Technology, holds a BA in English from the University of Minnesota and an MBA from St. Edwards University in Austin.

George Noren

IBM's Solution Partnership Center

By Suzanne Briggs

Powered by IBM's newly formed and fully dedicated Solution Developer Operations, solution providers now have access to the complete range of IBM platforms—from RISC hardware and software to OS/2 Warp products and beyond. This includes access to all aspects of these environments for testing and porting, in addition to assistance with go-to-market strategies.

Software vendors have long relied on IBM's AIXwest technical support facility to simplify their migration from non-IBM platforms to AIX. Renamed the Solution Partnership Center, the San Mateo, California-based facility's expanded services reflect IBM's intensified commitment to furthering the success of solution developers throughout the world. Its purpose is twofold:

- To help developers streamline the time-to-market process for applications using industry-leading IBM platforms
- To create and expand market opportunities for those applications

A second center will open this summer in Boston to serve East Coast software vendors, systems integrators, and consultants; more sites are slated to open throughout the year in Asia/Pacific, and Europe.

According to the Solution Partnership Center's manager, Patricia Meacham, the timing could not be better. "To be truly competitive in today's global marketplace, it has become essential for developers to provide client/server distributed solutions across the enterprise," she emphasizes. Since technology alone does not smooth this transition to client/server, Solution Partnership Center's support includes business seminars, marketing and vendor recruiting programs, product education, 24-hour porting labs, and on-site technical assistance—all free of charge.

Suzanne Briggs

Solutions for the Ongoing Evolution

As IBM technology evolves, so will the Solution Partnership Center. By the end of 1995, the center will support AS/400® and ES/9000™ products as well as networking and object technology. It currently offers free access to the following IBM platforms: OS/2® Warp™, the RISC System/6000 family including Symmetric Multiprocessing (SMP), AIX, POWERparallel™ (SP2™), Power Personal (PowerPC™), the DB2® family, and Client/Server products.

Eric Leong, a solution developer program manager for IBM, believes that today's growth path for applications is unparalleled. "In terms of scalability," he says, "the center gives solution providers the ability to concurrently enable products on the smallest PowerPC to the largest SP2, demonstrating the scalability across IBM's RISC platform." Leong goes on to illustrate how developers working in the AIX environment, for example, can come in to learn and work with the latest client/server and intelligent tools. These same hands-on opportunities also exist across the board for developers using leading-edge object technologies such as Taligent, System Object Model (SOM), OpenDoc, and human-centered products such as pen, speech, and multimedia.

Marketing, Recruiting, and Education

Helping solution developers identify the most effective and profitable channels for getting their products to market is one of the benefits of the Solution Partnership Center. "IBM is clearly committed to assisting developers with their marketing strategies," explains Meacham. "Our business seminars target expanding market opportunities, as well as issues shaping the computing industry and how they directly impact applications."

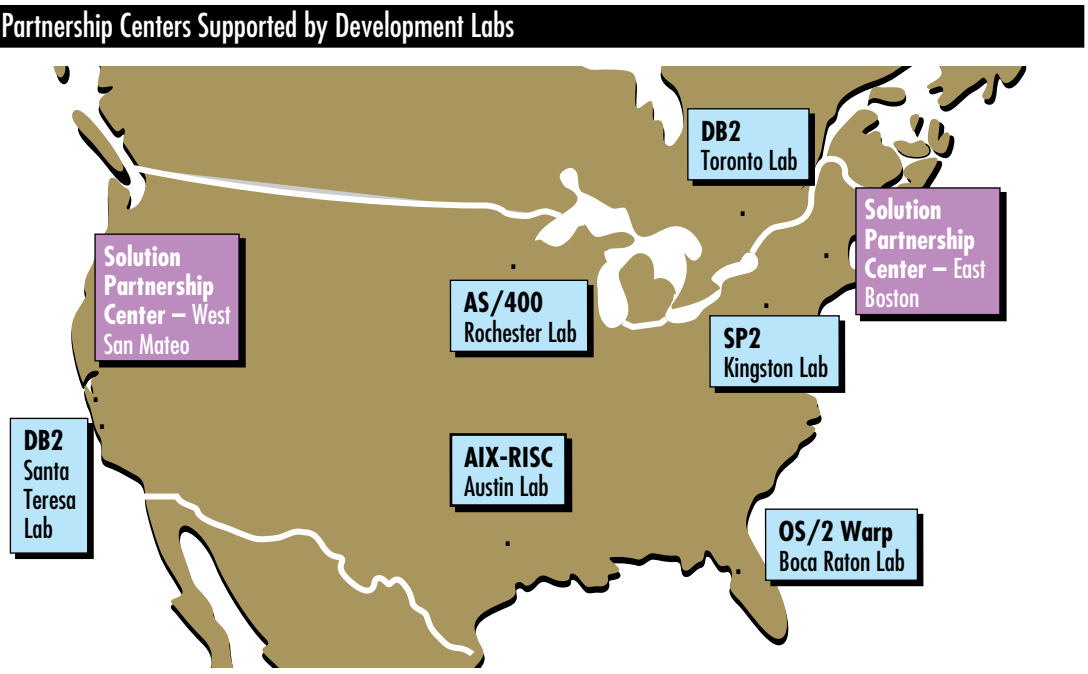


Figure 1. Solution Partnership Centers in the U.S. and Canada

The center's ongoing schedule of informative business seminars, consultant briefings, and product briefings also keeps solution developers on top of emerging IBM technologies, including Client/Server, PowerPC- and POWER-based systems, and software solutions for object technology, National Language Support (NLS), and communication products. Solution developers are privy to the latest product evaluations, publications, and a variety of special-interest sessions that cover topics such as bottleneck determination and isolation, implementing decision support on the SP2, and recruiting software partners.

IBM works with developers to forge mutually beneficial marketing partnerships through its Business Partner Program. This joint recruiting mission includes trade show opportunities, listings in IBM solutions catalogs, entry into IBM's National Solution Center database, and more.

Porting Convenience

Another advantage of the Solution Partnership Center is its 24-hour porting labs where solution providers can port and validate applications across IBM's entire spectrum of systems and configurations, evaluating performance and fine-tuning code. Users work at their own pace in a private, secure, and interruption-free environment that also provides benchmarking and prototyping assistance, and an expansive networking lab for product testing.

Whether investigating the feasibility of an application or conducting application development work, users are encouraged to exploit all the technology and information available to them through the labs. Access is scheduled in advance so the center can have a solution developer's specific environment set up and ready upon arrival.

Direct Technical Support

Even the most experienced solution developers need technical support now and then; and when they do, Solution Partnership Center is ready. Experienced technical consultants are available on-site, as needed, supported by product developers at IBM labs in Austin, Texas (AIX, RISC, SMP); Kingston, New York (SP1™ and SP2); Boca Raton, Florida (OS/2 Warp); Toronto, Canada (DB2); and other locations worldwide.

For more information about Solution Partnership Center services and how they assist in deploying and marketing IBM-driven applications, call 1-800-678-4249 in the U.S. From outside the U.S., call 415-312-0240.

"To be truly competitive in today's global marketplace, it is essential for developers to provide client/server distributed solutions across the enterprise."

Suzanne Briggs, IBM Corporation, 2929 Campus Drive, San Mateo, CA 94403. Ms. Briggs is the marketing communications manager at IBM's Solution Partnership Center—West. She has a BS in Mathematics and Computer Science from Georgia State University in Atlanta.

The Solution Developer Support Home Page

By Tom Hopkins

The World Wide Web is an excellent vehicle for providing timely information to broadly distributed groups. This article describes one of the efforts underway at IBM to take advantage of the Web to support solution developers.

The World Wide Web (WWW) had its beginning in 1989, when scientists at the CERN Research Center in Switzerland proposed a mechanism that would allow them to share work and disseminate information to the high-energy physics community worldwide. In January 1992, after several internal versions were developed, they made a line-mode browser available to the world via anonymous ftp. By early 1993, there were about 50 http servers in the world; and http traffic, the main protocol of the WWW, accounted for about 0.1% of the NSF backbone.

By September 1993, the traffic was up to 1% of the NSF backbone, and the National Center for Supercomputing Applications (NCSA) released working versions of the MOSIAC browser for common computing platforms; the world was about to change. The byte count for WWW traffic has jumped from 78 million in December 1992 to 225 billion in December 1993 to 3.5 trillion in December 1994 (see Figure 1). The Web is here!

World Wide Web

The WWW consists of interconnected documents in a variety of formats and a collection of programs that can understand the many information-retrieval protocols used on the Internet today. The WWW merges the techniques of networked information and hypertext to make an easy, but powerful, global information system.

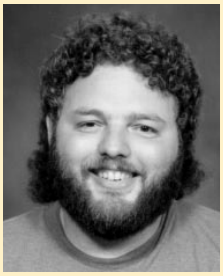
In using the Web, you look at a document, then point and scoot. Click on a link—anything from an underlined word to an icon to a place on a map—and you are off to a new document, or sound file, or image, or whatever is located in the U.S., or Germany, or Japan, or wherever. The possibilities are limitless.

In May 1994, IBM went live on the World Wide Web with a broad set of information designed to assist IBM's broad customer and partnership base. One of the driving forces behind the initial effort was the RISC System/6000 Division with its home page (http://www.austin.ibm.com), and under it, the POWER Team home page.

The goal of the POWER Team home page is to assist AIX solution developers by providing information about programs for AIX developers, details about AIX solutions, and a wide array of AIX technical information. Page usage has been strong from the beginning. Throughout the year, we added more and better information and responded to feedback. Although we were happy with what we were accomplishing, we thought that we could do so much more. Then in November, we started down the path to a WWW page that could provide that "so much more." Solution Developer Operations (SDO) was announced.

This new organization gives us the ability to combine our strengths and resources to create a page that will truly show IBM's commitment to solution developers to create one of the best offerings in the industry

The new SDO page (see Figure 2) is located at URL: http://www.austin.ibm.com/developer. It can also be accessed from many places in the IBM Web structure, including the Product, Service, and Support sections of the IBM home page (http://www.ibm.com). Since the Web is a living book that constantly grows and changes, this article will already be somewhat out of date by the time you read it. We plan to frequently add improvements based on the organizational foundation present today.



Tom Hopkins

The Information Architecture

The SDO page is a central point for developers on all IBM platforms and technologies. The top-level pages contain information of interest to developers who work on multiple platforms (see Figure 3); it also contains items from the topic pages presented in a cross-platform context. These top pages also include pointers to topic pages with a particular focus, such as AIX, OS/2, or object technologies. The SDO top page has the following sections:

- **Welcome to our server**: Gives a description of the who, what, and why of SDO as well as fast paths to other sections of the top page.
- **Developer news**: Represents the newspaper of the SDO page. Prominently displayed are key headlines with links to the full news item as well as pointers to a more comprehensive news page with current news and an archival section. There are also links to announcements from solution developers who are working with our technologies.

  Another part of the developer news section is "What's New," which lists documents added to the SDO page in reverse chronological order. A quick stop in this section during a visit will ensure that you do not miss something that has been added since your last visit.
- **Events and workshops**: Provides a calendar of upcoming conferences as well as worldwide workshop schedules. Watch this space for new features that will be implemented over the next six months.
- **Developer assistance from IBM**: Provides pointers to the various programs offered by IBM to assist you in developing, porting, or marketing solutions for IBM platforms. There is also information about facilities available to help you.
- **Information for solution developers**: Represents the core of the SDO page. A pointer to the SDO Main Library is the first part of this section. It points to a vast array of information from the individual topic page developer libraries such as technical papers, periodicals, and directories, as well as general developer documents including back issues of the *SDO Developer Support* newsletter. This section also includes pointers to a growing list of topic pages, each one with many levels of information in context for the topic. Closing out the area are pointers to the IBM home page and its IBM products and services page.
- **Other interesting places in cyberspace**: We will be watching for WWW sites that might be of interest to you. Luckily, adjoining

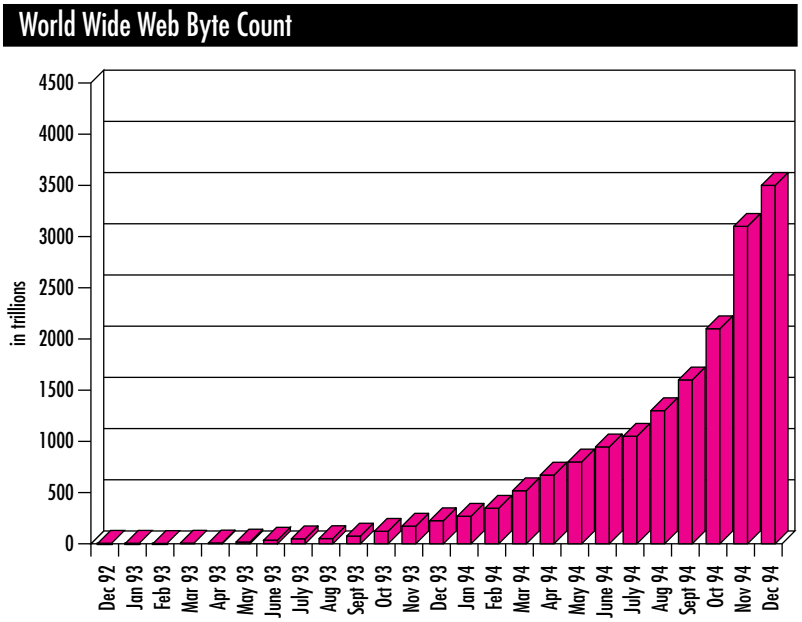


Figure 1. World Wide Web byte count on the NSFnet backbone

Figure 2. Solution Developer Operations home page

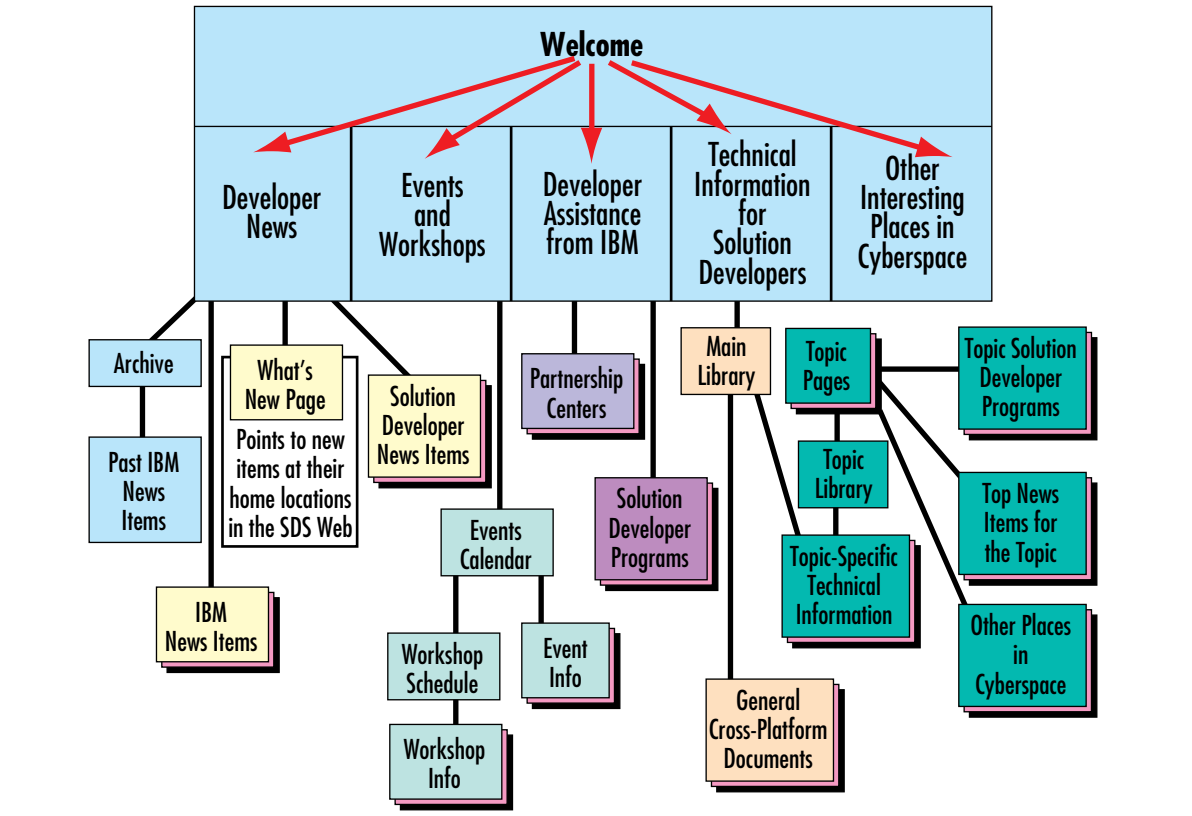


Figure 3. Main structure of Solution Developer Support home page

"interesting places" is a pointer to a form for you to send us feedback. Let us know what you think as well as what you would like to see added or expanded.

Navigation bars are located at the bottom of the SDO top page and also at the bottom of every document in the SDO page. The bottom bar is the same on every page and points to functions shared by the entire IBM Web community. The first bar, however, will change to suit the context of the document. It will start with a pointer to the SDO home page top and to the SDO Main Library, but may contain other items, such as a pointer to the Table of Contents (ToC) and Next page on compound documents, or a pointer to a topic page.

Topic Pages

Under the SDO home page is a series of topic pages with information for a particular set of solution developers. These pages are the home of general information and technical documents presented in the context of the topic. Although this article describes the AIX page in detail, you can expect to find similar types of data in the same place on most of the topic pages. You will also find that the structure parallels that of the top page. The following may be included on a top page:

- **Introduction**. Introduces the topic page.
- **List of top news items**. Ranges from product announcements to interesting news stories to summaries from recent conferences.
- **Developer programs**. Provides information about programs for AIX solution developers. If you have already looked at the programs from the top page, there is nothing new here. If you are only interested in the ones that touch AIX, there is information about POWER Team programs plus activities in North America and in Europe, as well as pointers to the IBM Developer Connection for AIX and the AIX Version 4 certification program.
- **Online library**. Contains technical information for AIX solution developers. The AIX topic page has pointers to key entries and a

link to the library itself. The library page contains many of the most popular documents from the old POWER Team home page. For example, Periodicals has the online version of *AIXpert* magazine including back issues to August 1993. Definitely watch this space! Directories has *AIX POWER Solutions*, containing over 4,000 entries of applications running on AIX. Other frequently accessed documents are the *All about AIX Version 4.1 Guide* (recently updated for 4.1.1) and the *AIX Version 4.1 Developer's Highlights* white paper. There is also a pointer to the AIX known-problems database.

Getting Connected to the World Wide Web (WWW)

After reading about it everywhere, you are now ready and eager to get connected to the Internet and surf the World Wide Web. So what do you do? First, get connected to the Internet; then locate a WWW browser for your platform.

Connecting to Internet

As an AIX user, you already have the basis for Internet connectivity—TCP/IP support in the operating system. With this type of support included, you only need to find an Internet access provider. One provider that offers everything from dial-up Serial Link Internet Protocol (SLIP) access to leased lines and firewall services is the IBM Global Network (IGN). For more information about IGN in the U.S., call 1-800-455-5056. If you already have Internet access, see URL: http://www01.ny.us.ibm.net/adv/ for more information.

If you choose the dial-up approach, you can find information for configuring SLIP in InfoExplorer and also in the 1-800-IBM-4FAX automated FAX server, available from IBM's WWW server (http://www.austin.ibm.com/) or under Services & Support if you have other means of connecting). *Internet World*[1], a monthly magazine published by Mecklermedia, is a good source for information about other Internet access providers.

For accessing the Internet from your personal computer, OS/2 Warp Version 3.0 is IBM's premiere operating system for PCs. It ships with a Bonus Pack of software containing an Internet Access Kit (IAK). This IAK provides a point-and-click interface for connecting to the Internet through IGN using SLIP; it can be used to connect to other access providers supporting SLIP and Compressed SLIP (CSLIP) or Point-to-Point Protocol (PPP). IBM also offers an IAK for DOS/Windows™ users. This IAK is available at local software stores.

Alternatively, many Internet access providers supply the necessary software for DOS PCs and Macintoshes® as part of their signup fee. New Internet access software packages arrive at computer retailers' shelves every day, as well as in the back of books at your local bookstores. Prodigy®provides Internet and WWW access through their Windows-based interface using their private network; other major online services have announced their intention to follow.

Locating a Browser

After connecting to the Internet, find a WWW browser (the interface for navigating the WWW) if one did not come with your Internet access kit. Most platforms have several available. The right one for you is primarily a matter of personal preference.

AIX/UNIX®, Macintosh, or DOS/Windows users can get the original NCSA MOSIAC by anonymous FTP from ftp.ncsa.uiuc.edu under the `/Mosaic` directory. For OS/2 users, IBM distributes Web Explorer by anonymous FTP at ftp.ibm.net in the `/pub/WebExplorer` directory. Web Explorer works with both the IAK distributed with OS/2 Warp and older versions of OS/2 with TCP/IP for OS/2 2.0. OS/2 Warp's IAK also supports most Windows-based browsers.

Once you can access the WWW, you can find information on other browsers at the following URL: http://www.yahoo.com/Computers/World_Wide_Web/Browsers/. After you start surfing, you will see why the WWW is one of the hottest topics in the media today. It brings a wealth of global information to your screen with an easy-to-use interface.—Ron Woan, IBM Corporation

[1] To request a subscription to *Internet World*, write to Internet World, P.O. Box 713, Mt. Morris, IL 61054, send E-mail to jwsubs@kable.com, or call 1-800-573-3602.

- **Links to other places**. There are also links to other places in cyberspace that an AIX developer might find of interest and the navigation bars which include a link to a list of POWER Team contacts worldwide.

This is only one example of a topic page. Other topic pages provide a wealth of information, including an excellent one about object technologies.

Futures

Other areas to be implemented in the coming months include the following:

- Adding information in all the topic pages
- Adding search facilities to certain areas, such as the *AIXpert* archive
- Expanding topic pages to cover subjects such as database and transaction processing
- Establishing a restricted area in which solution developers could access information and functions not available to the general public
- Improving the feedback form to make it easier for you to tell us what you want and what you think our priorities should be

Do not wait for the improved form. This home page is for you, a solution developer on IBM platforms. And we want your input for what you need. Contact us via the feedback form on the Solution Developer Operations home page or send E-mail to sdo@austin.ibm.com.

Other Waves on the Web

Check out these other IBM Web pages while surfing the Web:

- **http://www.ibm.com/** : The IBM home page; the heart of the IBM Web and the jumping off point for all IBM Web pages
- **http://www.austin.ibm.com/** : IBM RISC System/6000 home page; detailed product information about the RISC System/6000 family and AIX as well as home of technical papers
- **http://www.austin.ibm.com/pspinfo**.: IBM Personal Software; home for OS/2, PC-DOS, and LAN Server
- **http://www.austin.ibm.com/os2games**: Information for game developers as well as directories of OS/2 games and information about getting your favorite game to run on OS/2

Tom Hopkins, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Hopkins is the technical lead for the Solution Developer Support WWW project. He has a BS in Electrical Engineering from the University of Texas at Arlington.

NetView Association Services Extended to SystemView

In May 1995, IBM announced SystemView® Series (code name Karat), its solution for network and systems management. As part of the announcement, IBM unveiled a new program for SystemView that offers vendors the high-level of services provided by the NetView Association.

SystemView supports a variety of platforms and databases, offering the interoperability and connectivity required by users and a high level of access to data and applications anywhere in the network. And because SystemView is based on open industry standards, you can choose from a broad set of systems management applications.

SystemView marks the beginning of a new era of integration. From a single platform, users can manage any device or system in their enterprise. Management applications will have a common "look and feel," will share data, and will be able to invoke each other automatically to complete a task.

NetView Association members whose products run on any of the IBM NetView® platforms, such as AIX and OS/2, are automatically affiliated with the new vendor program.

For information about SystemView or about the new vendor program, send a note to nvassoc@vnet.ibm.com.

— *Larry Kunz*

IBM Marketing and Technical Services

By Wayne Mangelson

IBM is the largest software company in the world (based on total revenue), providing a distinctive number of products to end users and software developers. This article highlights technical, marketing, and business-related services for developers of OS/2 and AIX applications and tools.

IBM's wide array of marketing programs and services can enhance a software developer's success. With hot competition facing software companies and pressure to expand, success often requires selling applications on several platforms. Few companies can offer the breath, scope, and depth of services and products that are available from IBM.

Wayne Mangelson

IBM Products

- AD (Application Development)
- AIX
- Advanced Program-to-Program Communication (APPC)
- AS/400
- CICS™—MQ Series (transaction and messaging-queuing systems products)
- Communications Manager for OS/2 (CM/2)
- DB/2 (database)
- NetView
- Objects
- OS/2
- Pen
- Retail (Point of Sale)
- Speech Recognition
- System/390®
- Voice Type Dictation
- Work Group—Information Warehouse™

Figure 1. IBM products

IBM's Developer Assistance Program

IBM's software, hardware, and services constitute a total systems strategy. Solution developers can migrate from one product to another or create new tools and applications for different markets in conjunction with one company—IBM. The IBM exploitable products listed in Figure 1 illustrate this breath and depth of offerings.

Previously, each product in Figure 1 had its own unique set of developer support services and its own Developer Assistance Program (DAP). IBM is now restructuring these multiple DAPs and streamlining the many services available for developers to make it easier to do business with the company.

The recently announced Solution Developer Operations (SDO) Unit within IBM underscores this effort and brings together, into one group, all support for developers exploiting or writing applications and tools for AIX and OS/2.

The remainder of this article highlights these AIX and OS/2 developer services.

AIX POWER Team

The POWER Team helps solution developers access the IBM tools needed to grow their products in the AIX marketplace by using IBM's POWER Architecture™ and the RISC System/6000. The POWER Team is intended for developers working on or marketing products for commercial release. Any developer creating products for AIX or marketing AIX or UNIX products can apply for membership in the POWER Team.

The POWER Team services include both no-charge and fee-based technical, marketing, and business-related offerings. Figure 2 shows the support services for members of the POWER Team.

For more information or to request a membership application, call 1-800-627-8363 within the

POWER Team Support Services

Technical Support

- Porting assistance
- Porting information and white papers
- Ongoing technical support
- Consulting services (design reviews, performance tuning, on-site consulting)
- Benchmarking
- Information over the Internet and commercial online services (technical, marketing, E-mail, bulletin boards, forums)
- Question and answer support
- Early product information

Business Services

- IBM relationship or advocate representatives for the developer
- Hardware discounts, loaners, leasing, rentals, and purchase options
- Remote access to a RISC System/6000
- Newsletters
- Software discounts
- Access to AIX public-domain software

Marketing Services

- Entry in *AIX POWER Solutions* directory
- Registration in IBM's National Solutions Center (accessed by IBM's national sales force)
- *AIXpert* magazine subscription (quarterly technical publication)
- Advertising discounts in the *AIX POWER Solutions* directory and *AIXpert* magazine
- Shows and conferences (eligibility based on criteria established for each show)
- Direct mail
- Opportunity to use IBM Direct (IBM's toll-free 800 number sales channel)
- Consideration to use IBM's Application Solution Centers for customers and developers

Figure 2. POWER Team support services

U.S., 404-835-9902 outside the U.S. and Canada, or send E-mail to ibmspsc@austin.ibm.com.

OS/2 Worldwide Developer Assistance Program

The IBM Worldwide Developer Assistance Program is open to all developers of IBM personal software-based products (OS/2, OS/2 for the PowerPC, PC-DOS, Pen, multimedia, and LAN systems). IBM has several extensions that offer customized services for specific developer needs. Some services vary by country.

Worldwide DAP services are available to developers of personal software-based products, but also include corporate programmers producing in-house applications, MIS professionals, consultants, educators, industry analysts, government agencies, individuals with no company or product prerequisites, and others who have an interest in IBM's personal software-based products.

To worldwide DAP members, IBM can provide a variety of technical, business, and marketing support services, such as the following.

- Technical support through IBM OS/2 forums on CompuServe®
- Opportunity to participate in early-code programs
- Access to the OS/2 custom application porting center
- Information about technical conferences
- Product announcements

You can enroll electronically through CompuServe or the Internet as follows:

- Using CompuServe, enter GO OS2DAP and complete the online application form.
- On Internet, visit the IBM Solution Developer home page at URL: http://www.austin.ibm.com/developer.

Inquiries about the Developer Assistance Program can be directed via fax or voice telephone contact to the countries listed in Figure 3.

Geographic Area	Fax/Voice
Australia	61-2-354-7766 (fax)
Europe, Middle East, and Africa	44-0-1256-336778 (fax)
Japan	81-3-3279-8231 (fax)
Latin America and South America	525-627-2086 (fax)
Taiwan	886-2-752-1577 (fax)
United States and Canada	407-998-7610 (fax) 407-982-6408 (voice)

Figure 3. Developer Assistance Program contacts

Additional Services

Developers in the U.S. and Canada may be eligible for additional services. If you develop or market applications or tools for commercial release and base those products on IBM personal software, you may qualify for these services. A sample of these services includes the following (fees may apply):

- Online technical support at no charge
- Defect support for all warranted products that are supported by IBM support centers
- Complimentary access to the online database of personal software-based development tools
- Complimentary subscriptions to *OS/2 Developer* magazine
- Discounts on IBM personal software products
- Relationship (advocate) representatives
- Product compatibility program (IBM compatibility mark issued to vendors with products that are compatible with IBM's platforms)
- Directories for advertising and listing solution applications
- Advertising programs
- Bulletin board/Internet information posting and exchange
- Shows and conferences (participation based on criteria established for each show)
- User groups
- Press announcement assistance
- Direct mail support
- Sales through IBM's 800 number channel (1-800-3IBM-OS2 is a direct response channel called the Software Store for end users and independent software vendors interested in OS/2 applications)
- *DSNews* newsletter
- Customized marketing planning assistance

Wayne Mangelson, IBM Corporation, Internal Zip 4105, 11400 Burnet Road, Austin, TX 78758. Internet: wayne_mangelson.ausnotes@aussmtp.austin.ibm.com. Mr. Mangelson develops marketing programs to help AIX and OS/2 solution developers successfully market their products. He has a BA in Economics and Statistics and an MS in Administration and Organization Psychology from Brigham Young University in Provo, Utah. He also has a PhD from the University of Michigan.

Developer Support Newsletter

DSNEWS is published electronically and monthly by IBM's Solution Developer Operations Unit on several E-mail and BBS systems.

To read recent issues online, go to the Internet Solution Developer Support home page at URL: http://www.austin.ibm.com/developer. Look in the Periodicals section of the home page main library.

Issues can be downloaded from the following services:

Internet
Anonymous ftp from software.watson.ibm.com in directory/pub/os2/info

CompuServe
OS/2 form in Newsletters library

America Online®
OS2DF2 forum, *DAP library section 14

Prodigy
OS/2 club topics download library in IBM files

IBM
ASCII format in DSN5 PACKAGE of OS/2 Tools catalog

The IBM Developer Connection for AIX

By Charlie Cree and Syed Z. Pasha

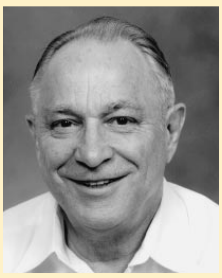
The IBM Developer Connection for AIX is a subscription program to support developers of AIX software. It offers streamlined access to cutting-edge IBM development information and technology. The program enhances your ability to produce more high-quality AIX applications.

In the market for AIX application software, it is almost axiomatic that the winner is the first one there. Because the demand for AIX program products is so strong, customers generally buy quality new products quickly, leaving little volume for those late to the party. DevCon can help *you* be the winner.

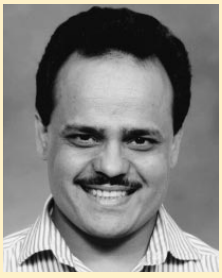
The IBM Developer Connection for AIX

The IBM Developer Connection for AIX (called DevCon) is a subscription program designed to enable developers to bring quality products to the marketplace in a timely manner by providing them with leading-edge IBM tools, technology, and know-how.

DevCon is a 12-month subscription with up to four issues during the subscription period. Each issue includes updates and additions to previous issues. The subscriber receives hypertext AIX documentation, prerelease products, development kits, development tools, papers and books, sample code, demonstration versions of programs, magazines, newsletters, and trial programs.



Charlie Cree



Syed Z. Pasha

Packaging and Content

Although some printed matter (especially the latest issue of the newsletter) is distributed with each issue, the primary content is distributed on two or more CD-ROMs. One containing hypertext libraries is usually distributed only once per subscription. Other information is distributed on additional CD-ROMS, which are part of every issue. For example, the second issue (Volume 2, Issue 1) contained the following:

- Hard copy of the newsletter
- Hypertext CD-ROM for subscribers
- DevCon CD-ROM containing the following:
  - Copies of newsletters
  - White papers about the POWER2 and PowerPC microprocessors
  - OpenDoc and architectural papers dealing with fixed- and floating-point arithmetic, performance, migration, and compatibility
  - Various books, including several on Distributed Computing Environment (DCE) and Encina®, as well as the *AIX 4.1 Porting Guide*
  - Issues of *AIXpert*
  - Tools such as AIX MOSIAC, the Mathematical Accelerator Subsystem (MASS), the AIX Desktop Graphical User Interface (GUI) Builder, and the AIX Font Selection Tool
  - System administration and productivity tools
  - Sample device drivers for PCMCIA, ISA, and Micro Channel®
  - Prerelease versions of REXX, Common Desktop Environment (CDE), and Program Visualization
  - Demo programs, such as the suite of AIX windows programs and the Performance Toolbox

System Requirements

The system units must support AIX Version 4.1 or higher. Although a minimum of 16 MB of memory are required, 32 MB are recommended. A CD-ROM drive supported by AIX is required. The disk space requirements vary, depending on the software installed and the user data. Other requirements may be dictated by the specific program being executed.

Prices

The IBM Developer Connection for AIX subscription is $495 (U.S.) per year. Qualified developers can receive a discounted price of $250 (U.S.) per subscription under the RISC System/6000 Developer's program or under the Power Personal Developer's toolbox. These prices may change without notice.

Developer discounts are also available to students and faculty of certain educational institutions.

Advantages of the DevCon Solution

The DevCon solution streamlines access to cutting-edge IBM development information and technology. This helps developers to shorten the development process in the following areas:

- Using the best tools and technology available to develop or port program products
- Devoting fewer resources to writing placeholder or contingency code
- Reviewing the technology, information, and tools delivered at their doorstep

How to Become a Qualified Developer

To obtain a qualification kit, developers in the U.S. should call 1-800-627-8363, ext. 25, or fax a written request to 404-835-9444.

Developers in other countries should contact their local IBM marketing representative to determine the developer discount structure and qualification criteria in their respective countries.

How to Subscribe

The IBM Developer Connection for AIX is available worldwide. Subscriptions can be ordered by calling the telephone numbers in Figure 1.

In addition, telephone numbers are provided in Denmark for the convenience of customers who do not live in the countries named in Figure 1. Multi-lingual operators can be reached in Denmark at the following numbers:

Language	Telephone
Dutch	45-48101400
English	45-48101500
French	45-48101200
German	45-48101000
Italian	45-48101600
Spanish	45-48101100
Scandinavian languages	45-48101300

In the U.S., the normal order-processing time for the first issue is 48 hours after receiving the order. Subsequent issues that are part of the subscription are sent out automatically.

Country	Telephone
Australia (covering the Pacific area)	61-2-354-7684
Brazil	021-800-6120
Canada	1-800-561-5293
Mexico City	627-2444
Mexico	91-800-00639
United States	1-800-6DEVCON (1-800-633-8266)

Figure 1. Numbers to call to subscribe to DevCon

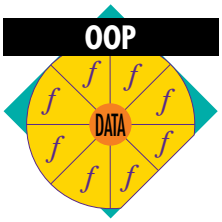
Support for Developers

Currently, the following forms of support are available to DevCon subscribers:

- For technical support or questions, send a note via E-mail to devconAIX@austin.ibm.com
- To obtain Program Temporary Fixes (PTFs), use the following Internet address: ftp://software.watson.ibm.com

Charlie Cree, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Cree is responsible for AIX application development, application development tools, and object technology tools requirements and strategy.

Syed Z. Pasha, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Pasha manages the DevCon for AIX project. He is responsible for AIX application development, database and high-performance computing requirements and strategy.



Distributing Objects with Orbix

By Annrai O'Toole

Object Request Broker (ORB) technology is an essential component in developing distributed object-oriented applications. This article discusses the Common Object Request Broker Architecture (CORBA) using examples from Orbix, a commercially available implementation of the CORBA specification from IONA Technologies.

The Object Management Group's® (OMG's®) Common Object Request Broker Architecture (CORBA) is a viable option for attaining open and interworking object-oriented software. CORBA is a set of rules for developing software objects and applications that will interoperate across many system platforms and operating systems.

Several vendors are already providing implementations of CORBA specifications: IBM with Distributed System Object Model (DSOM), Digital, HP™, Sun®, and IONA Technologies. IONA began shipping Orbix, the first complete CORBA implementation with a C++ binding, in June 1993.

The basic CORBA specification published by OMG describes an Interface Definition Language (IDL) and the relationship between the IDL and C, C++, and SmallTalk®. The specification also lists several standard interfaces that must be provided in all ORB implementations. OMG is currently engaged in a process to provide a specification for an IDL to Ada mapping. Other language bindings are expected.

IDL defines interfaces, not how the interfaces are actually implemented. It provides constructs for defining new types, including complex types such as arrays, sequences, structs/records, and unions/variants, as well as the interfaces. The operations supported by interfaces help to define the interface, and inheritance relationships between interfaces can be defined. IDL does not provide executable statements—there are no assignments or statements such as `if`, `while`, or `for`.

IDL makes CORBA an open standard. It defines interfaces that are neutral to any particular programming language. Interfaces to objects can be defined in IDL; they are then automatically transformed into the corresponding interfaces in different programming languages. For example, a C++ client could use a Smalltalk object and vice versa; objects may also be programmed in non-object languages such as C, COBOL, or Ada.

A Simple CORBA Application

Consider a simple banking application in which entities such as banks, bank accounts, and bank clerks can be represented as objects. This example shows inheritance—because there can be different categories of accounts. It also shows distributed programming—because a customer may want to access a bank account remotely from the bank server machine that maintains it.

Similar to C++, the account interface has three components: a read-only attribute that represents the current balance in the account, and two operations to alter the balance. For simplicity, we represent sums of money as floating-point numbers, although this would not be suitable in a commercial application because of potential numeric rounding errors. Figure 1 shows how the basic functionality of a single bank account may be expressed using IDL, and the code produced when an IDL compiler translates it into C++.

The attribute balance has been represented in C++ as a member function returning the value of the balance. If the attribute had not been read-only, there would have been a second member function—taking a float argument and returning a void—to set the balance.

IDL can be translated into C++, but it is generally not possible to translate all C++ type


Annrai O'Toole

declaration constructs into IDL. For example, IDL does not allow overloading of member functions (constructors or destructors), or default and ellipsis arguments. Since IDL is language neutral, any features supported by IDL should be translatable to a wide range of popular object-oriented and non-object-oriented languages.

Both account objects and bank objects to manage the accounts are required. Figure 2 shows the bank interface using IDL and the interface mapped into C++.

References to objects, such as accounts described in IDL, are represented as pointers to classes in the C++ equivalent—shown here as `account*`.

The IDL compiler also builds definitions of the member functions of account and bank classes; each of these definitions builds and sends an appropriate request to a remote server.

Example Client Program

A major benefit of CORBA is its ease in building distributed programs. The following example shows how to use the account and bank classes in a client program that executes remotely from the bank server program, which in turn manages the bank and maintains the accounts.

Figure 3 shows C++ code examples using several specific features of Orbix. ORBs from other vendors provide similar functionality.

Orbix provides the C++ static member function `_bind` call that requests the ORB to search the network for any bank server, activating such a server if necessary. Figure 4 compares a search of the network for a specific bank server to a search of a specific host ("Bank" refers to the name of the bank).

Although header files are ignored here, in practice, the program would include both `<stream.h>` and the C++ header file generated by the IDL compiler containing the C++ equivalent of the bank and account IDL interfaces. Possible error conditions are also ignored in these examples.

The IDL exception-handling mechanism allows standard ORB errors to be described, and also allows object programmers to introduce their own application-specific exceptions. Any IDL-described operation can result in an exception. Exceptions are mapped in C++ as a trailing, defaulted argument to every operation. In fact, the C++ code generated by the IDL compiler for an account is shown in Figure 5.

Because environments are defaulted, they can be ignored when appropriate; that is, when the

Bank Account Expressed in IDL

```
//IDL
interface account {
   read-only attribute float balance;
   void makeLodgement (in float f);
   void makeWithdrawal (in float f);
};
```

Bank Account Translated into C++

```
// C++
class account {
public:
   virtual float balance ();
   virtual void makeLodgement (float f);
   virtual void makeWithdrawal (float f);
};
```

Figure 1. The account IDL interface and its mapping in C++

Bank Interface Using IDL

```
//IDL
interface bank {
   account lookUpAccount (in string customerName);
   account newAccount (in string customerName);
};
```

Bank Interface Mapped into C++

```
// C++
class bank {
public:
    virtual account* lookUpAccount (char* customerName);
    virtual account* newAccount (char* customerName);
};
```

Figure 2. Bank IDL interface and its mapping in C++

```
// C++
main () {
   // bind to *any* bank service
   bank *b = bank::_bind ();

   // create a new account
   account *a = b->newAccount ("Marie");

   // lodge $10
   a->makeLodgement (10.00);

   // output the balance
   cout << "balance is " << a->balance () << endl;
}
```

Figure 3. A simple client application

Searching the Network

```
bank *b = bank::_bind ("Palo Alto:Bank");
```

Searching a Specific Host

```
bank *b = bank::_bind ("Palo Alto:Bank", safeHaven.mil);
```

Figure 4. Controlling what interface object a client application use

```
// C++
class account {
public:
  virtual float balance
     (Environment&env=default_environment);
  virtual void makeLodgement
     (float f,Environment&env=default_environment);
  virtual void makeWithdrawal
     (float f,Environment&env=default_environment);
};
```

Figure 5. Adding CORBA environments to operation signatures

```
// C++
main () {
   TRY {
     // bind to *any* bank service
     bank *b = bank::_bind (IT_X);

     // find an account
       account *a = b->newAccount ("Marie",IT_X);

     // lodge $10
     a->makeLodgement (10.00,IT_X);

     // output the balance
     cout << "balance is " <<
              a->balance (IT_X) <<endl;
  }
  CATCHANY {
     cerr << "Unexpected exception "
            << IT_X << endl;
  }
  ENDTRY
}
```

Figure 6. Adding exception handling to a CORBA application

client and server object are on the same machine or in the same UNIX process. In these cases, there are no exceptional conditions caused by network or machine failure. All predefined identifiers relating to the ORB are scoped with an enclosing class module called CORBA, resulting in the following environment argument (the CORBA scoping is omitted in the code below).

```
CORBA::Environment&
   env=CORBA::default_environment
```

Figure 6 shows several predefined macros in Orbix that simplify catching an exception in the client program.

The TRY macro introduces the (automatic) variable `Environment IT_X`. The Environment class supports `operator <<` on ostreams, so the details of the error can be simply generated.

In Fall 1994, support for C++ exceptions was added in the OMG C++ to IDL mapping adopted by the OMG. This C++ forms part of the CORBA 2.0 specification and will be available from IONA in mid-1995.

Example Server Program

The server program implementation is straightforward. In addition to generating the account and bank C++ classes used by the client programs, the IDL compiler also declares two derived classes of these for use by server programmers, as shown in Figure 7.

Each class inherits from the corresponding C++ class that is also used by the client code. Because one server can be a client of another, it requires access to the code to construct and marshal requests.

The member functions of the `BOAImpl` classes are defined `=0` so that these classes are abstract and cannot be directly instantiated. Server programmers are expected to implement each member function in a derived class. Figure 8 shows implementations of the account and bank classes.

The list of opened accounts is maintained in a serial list. Figure 9 shows how to build a new account.

The `acc` is a derived class of account. We have ignored certain rules used by the ORB and server code for managing dynamically allocated memory. When a request for a `newAccount` arrives, the ORB dynamically allocates a new string for the customer name argument and deletes this string when the request has finished. For this reason, it is important that the server program return a copy of the string to the client application.

```
// now build new account
   account *pa = new acc
      (strdup(customerName), 0.0);
```

Similarly, the ORB recovers the storage occupied by any IDL-defined objects when these are passed to it as return or out parameters of a

request. To avoid losing the newly created account when a pointer to it is returned from the function, the ORB must be instructed to keep it after the request finishes:

```
// now build new account
   account *pa = new acc
      (strdup(customerName), 0.0);
   pa->_duplicate ();
```

The `_duplicate` increments a reference count associated by the ORB with the account object. The ORB will delete the object once its reference count reaches zero.

The following represents the server mainline:

```
main () {
   banker myBank;
   Orbix.impl_is_ready ();
}
```

The mainline is clearly trivial. Incoming requests are directed to a banking object, telling the ORB that we are now ready to take incoming requests. The server is implementing two distinct IDL interfaces: bank and account. The mainline needs to create only a bank; the bank can create accounts as requested.

The `impl_is_ready` call will automatically timeout if consecutive requests are not received within a certain period (approximately five minutes). The timeout value can be explicitly specified in milliseconds:

```
Orbix.impl_is_ready (30*60*1000);
   // 30 minutes
```

Like most calls on the ORB, `impl_is_ready` can raise an exception; for example, the ORB itself cannot find its configuration information. If the `impl_is_ready` eventually times out, it does not raise an exception; timing out is considered the normal outcome of `impl_is_ready`. The following is a more appropriate way to code the mainline:

```
main () {
  banker myBank;
  TRY {
     Orbix.impl_is_ready (IT_X);
  }
  CATCHANY {
     cerr << "Exception:: " << IT_X <<
        endl;
  }
  ENDTRY
}
```

Account Skeleton Class

```
// C++
class accountBOAImpl : public account {
public:
   virtual float balance
       (Environment&env=default_environment)=0;
   virtual void makeLodgement
       (float f,Environment&env=default_environment)=0;
   virtual void makeWithdrawal
       (float f,Environment&env=default_environment)=0;
};
```

Bank Skeleton Class

```
class bankBOAImpl : public bank {
public:
   virtual account* lookUpAccount (char* customerName,
         Environment&env=default_environment)=0;
   virtual account* newAccount (char* customerName,
         Environment&env=default_environment)=0;
};
```

Figure 7. Skeleton classes for bank and account

Account Class

```
class acc : public accountBOAImpl {
   char    *m_name;
   float m_balance;
public:
   acc (char* name, float bal) { // constructor
      m_name = name; m_balance = balance; }

   float balance (Environment&env=
                  default_environment) {
      return m_balance; }

   void makeLodgement (float f,Environment&env=
                   default_environment) {
      m_balance += f;}

   void makeWithdrawal (float f,Environment&env=
                    default_environment) {
      m_balance -= f;}
};
```

Bank Class

```
class banker : public bankBOAImpl {
protected:
   struct accountsList { account   *ac;
      accountsList   *next;
      };
      accountsList   *m_head;

public:
   virtual banker () {m_head = nil;}
   account* lookUpAccount (char* customerName,
        Environment&env=default_environment);
   account* newAccount (char* customerName,
        Environment&env=default_environment);
};
```

Figure 8. Implementation classes for bank and account

```
account* banker::newAccount (char* customerName,
                        Environment&) {
   // check customer's account does
   // not already exist...
   // now build new account
   account *pa = new acc (customerName, 0.0);

   // insert it into the list...
   // finished
   return pa;
}
```

Figure 9. Implementing the newAccount operation

The ORB knows that incoming bank requests should be directed to `myBank`. What would happen if the mainline created several different bank objects—which one of them would be used? These issues are discussed in the next section.

Activation

The ORB uses a registry of servers called the *implementation repository*. Servers must be preregistered with this repository before they can be used with an ORB. Once registered, the ORB will automatically relaunch that server if any client attempts to use it. The ORB also allows the server to be launched manually.

A shell-level command registers a server as follows:

```
% putit bank /usr/users/me/a.out
```

The `putit` command and the implementation repository associate a server name with an executable file. Server names must be unique within a repository, but they can be hierarchically structured. Each server machine node in the network is expected to have access to at least one repository. Different server machines can use different repositories.

Th following example uses the server name Bank:

```
% putit Bank /usr/users/me/a.out
```

When a client does a `bank::_bind()` request, the ORB uses the implementation repository to find a server named bank (and not Bank) If the server name differs from the IDL interface name (Bank differs from bank), then the `_bind` request must name the server explicitly, and the ORB will search the repository for a server called Bank, not bank:

```
bank *b = bank::_bind (":Bank");
```

A client program can also specify a specific object within a server to which it wishes to bind:

```
bank *b = bank::_bind ("Palo Alto:Bank");
```

In this case, the server must name the objects to which it expects clients to bind, using the `_marker` call:

```
main () {
  banker myBank;
  myBank._marker ("Palo Alto");

  banker anotherBank;
  anotherBank._marker ("Walnut Creek");

  TRY {
    Orbix.impl_is_ready (IT_X);
  }
  CATCHANY {
    cerr << "Exception:: " << IT_X <<
       endl;
  }
  ENDTRY
}
```

If a client does not specify a particular object within a server during a bind, then the ORB can choose any type-compatible objects. If a client enters `bank *b = bank::_bind ();`, then either `myBank` or `anotherBank` can be used to satisfy the bind.

Execution Trace

The bank server creates a banker object when it is run. This results in a quiescent server, waiting for incoming requests.

```
main () {
  banker myBank;
  TRY { Orbix.impl_is_ready (IT_X);
```

When the client starts, it first binds to any bank service:

```
main () {
   // Bind to *any* bank service
   bank *b = bank::_bind()
```

The result of the binding is an automatically generated proxy or surrogate object that acts as a stand-in for the remote banker object in the server.

Programmers do not need to be aware of the proxy object since it is managed automatically by the ORB. The client program then asks the bank to open a new account.

```
// find an account
account *a = b->newAccount
   ("Marie",IT_X);
```

Within the server, `banker::newAccount` is called, generating a new `acc` object. The `acc` object is linked to the banker object's list of accounts (via `banker::m_head`). Finally, `newAccount` returns a pointer to the account back to the client. At the client side, a new proxy for the remote pointer is automatically created.

Collocation

For some applications, IDL can be used to define the interfaces between components, even if these components are not distributed. Components of some applications may be distributed, depending on how the application is configured by its installer. Developers of distributed applications may want to temporarily run code with all components bound in the same process address space, such as to aid debugging.

To support these issues, the ability to collocate client and server objects within the same process address space is essential. For collocation to be as efficient as normal object invocations, the underlying ORB should not normally be involved in the invocation path once a binding has been established. In particular, messages should not necessarily be marshaled into and unmarshaled from message buffers. This implies that the mapping from IDL to the target programming language (such as C++) should be perfectly symmetrical between the client and server sides of each interface. Consequently, client and server code can be directly bound together.

Orbix supports collocation by linking with a particular version of the ORB library, which never attempts to locate an object outside the current process address space. A program can be relinked to use the normal form of the ORB library; a runtime test can also be made to determine the library to which the application is currently linked.

If collocation is always to be used for a particular (non-distributed) application, it may be appropriate to ignore the trailing environment values in client calls, thus ignoring system-raised exceptions.

Figure 10 shows a collocatable form of the bank example.

In effect, the initialization of the server mainline has been moved into the client code (a heap allocated object was used instead of an automatic object, which retains the server object after the collocation test). The code in Figure 10 can then be run collocated or distributed (against the original server code). In the collocated case, an execution trace would result in dynamic invocation.

```
// C++
main () {
  // build server object in this process
  // if need be...
  if (Orbix.collocated ()) {
     banker *myBank = new banker (); }
  // bind to *any* bank service
  bank *b = bank::_bind ();
  // create a new account
  account *a = b->newAccount ("Marie");
  // lodge $10
  a->makeLodgement (10.00);
  // output the balance
  cout << "balance is " << a->balance () <<
endl;
}
```

Figure 10. Collatable bank example

Dynamic Invocation

IDL describes the interfaces to an object. In the previous application examples, we examined how to write an IDL interface and then provide some implementation code behind that interface. We showed how to build client code to access that IDL object. The client application used code generated by an IDL compiler. All the code to build and send a request to the IDL interface was compiled statically into the client application at compile time.

Using the IDL compiler in this way limits some applications. The IDL interfaces that a client program can use are determined when the client program is compiled. Therefore, the client code can use only those servers that provide the IDL interfaces selected by the client programmer when building the application. Some application programs and tools require that they can use an indeterminate range of interfaces—such as browsers, management support tools, command-line and interactive interfaces, and interfaces that perhaps are not even conceived at the time the application is developed.

The OMG's CORBA specification also introduces a dynamic invocation interface that allows an application to issue requests for any interface, even if that interface was unknown when the application was compiled. Invocations can be constructed at runtime by specifying the target object reference, the operation name, and the parameters. Invocation is dynamic, because the IDL interfaces used by the client application do

```
TRY {
  r.invoke (IT_X);
}
CATCHANY {
  cerr << "Exception " << IT_X << endl;
  // maybe exit
}
ENDTRY

Object *a;
r >> a;   // new account
```

Figure 11. Simple dynamic invocation

```
//IDL
interface account {
   readonly attribute float balance;
   void makeLodgement (in float f);
   void makeWithdrawal (in float f);
};

interface checkingAccount : account {
   readonly attribute float overdraftLimit;
};

interface bank {
   account lookUpAccount (in string customerName);
   account newAccount (in string customerName);
   checkingAccount newCheckingAccount
         (in string customerName, in float limit);
};
```

Figure 12. Interface inheritance using IDL

not have to be statically determined at the time the program is compiled.

A server receiving an incoming request does not know—or care—whether the client that sent the request used the static or dynamic approach to compose a request. However, the underlying ORB infrastructure must ensure that the request sent by the client is type-safe against what the server expects. Dynamically composed requests must be tagged at runtime with sufficient type information to enable the ORB at the server side to assert type safety at runtime. Clearly, the expense of runtime type assertions should only be incurred for dynamically composed requests.

The CORBA specification provides an Application Programming Interface (API) to allow client code to dynamically compose requests, based on lists of "named values." Each named value is also tagged with type information. A named value list (an NVList) constitutes the arguments, including `out` parameters, to any particular invocation.

CORBA Object Reference

A CORBA object reference (as opposed to a C++ pointer to a C++ object) is defined in CORBA by the IDL interface object. CORBA's object interface is also the implicit root of all other IDL interfaces. For example, both previously discussed IDL interfaces—account and bank—are implicitly derived from Object.

Any CORBA object reference can be translated into a character-string format by using the operation `object_to_string` (invoked on the ORB itself), and back again using `string_to_object`. This is one way in which an object reference can be initialized. The CORBA specification does not directly address obtaining suitable string values that can be converted into object references. For example, they could be obtained via an external name service into which object references had previously been registered.

All CORBA-conformant ORBs support the dynamic invocation API. Because the API is complex to use, Orbix also provides a "veneer" over the API in C++, which makes the API easier to use. In the bank example, the first issue is to build an object reference for the bank server. If we use `_bind` again, this assumes that we know in advance to use the bank interface and can use the `_bind` call generated by the IDL compiler from the bank interface. In general, we do not know in advance what interface we want to use. Instead, we might initialize an object reference from a character string obtained externally to the program, using the C++ constructor for Object[1].

We can now continue with a code example to demonstrate making the first dynamic invocation that will create a new account object. We first create a CORBA request object, then initialize it with the desired target object reference and operation name.

```
// create an account....
Request r (&b, "newAccount");
// strictly,CORBA::Request
```

Next, we insert the arguments. In Orbix, class Request supports a stream-based interface to do this: `r << "Marie"`.

Because C++ is strongly typed, the stream operators can tag the underlying NVlist with the correct type information. In contrast, the C

[1] Object is strictly `CORBA::Object`, but we have omitted `CORBA::` prefixes for brevity.

language interface to the dynamic invocation interface requires programmers to supply their own type tags. It is then possible to invoke and obtain the invocation result—in this case, another object reference for the newly created account, shown in Figure 11.

An exception will be raised during the invocation if the dynamic type checking fails. We can continue to further invocations.

For an IDL-specified attribute such as balance, the operation name used during the dynamic invocation interface must specify whether the value is being sought or set. The example above shows the dynamic approach. It is typical practice to require information about what IDL interfaces a particular (remote) object supports, and to compose messages to it. CORBA specifies further interfaces that assist in this work, specifically a service responsible for managing information about interfaces themselves—the interface repository.

The CORBA specification contains more information about the interface repository.

Inheritance in IDL

Inheritance in IDL allows a new interface to be refined by extending the functionality provided by the earlier one. The new interface inherits or derives from the first. This is similar to the way in which a C++ class can be derived from its base class. Both IDL and C++ support multiple inheritance, allowing an interface (or class) to have several immediate base interfaces (or classes).

Figure 12 shows an example of single inheritance using the bank account.

The new interface `checkingAccount` derives from `account`. We have also added a new operation to interface bank to manufacture `checkingAccounts` (we could also have derived from the interface bank to produce a new interface that supports the new operation; however, this might make the example more difficult to explain).

The C++ class hierarchy produced by the IDL compiler ignores the Environment arguments, shown in Figure 13.

The usage from a client is similar to the previous example. The new server code that results is also predictable from the earlier example (see Figure 14).

Conclusion

The banking example describes what using an ORB might look like from C++, following the specifics of the Orbix ORB, but using the general principles implemented by all CORBA-conformant ORBs.

Writing distributed applications using ORB technology does not require much additional learning once you understand C++. Using IDL to describe interfaces to system components (whether or not they are distributed) and using an ORB to bind components together can offer substantial savings in design, coding, and maintenance.

The CORBA specification provides a baseline technology for distributed object-oriented systems in the computing industry. OMG is currently considering various value-added services that can be standardized within COBRA and used to augment a basic ORB. Services being considered include object life-cycle and relationships, persistence

```
// C++
class account {
public:
   virtual float balance ();
   virtual void makeLodgement (float f);
   virtual void makeWithdrawal (float f);
};

class checkingAccount : public virtual account {
public:
   virtual float overdraftLimit ();
};

class bank {
public:
   virtual account* lookUpAccount (char* customerName);
   virtual account* newAccount (char* customerName);
   virtual checkingAccount* newCheckingAccount
           (char* customerName, float limit);
};
```

Figure 13. C++ mapping for IDL inheritance

```
// C++
main () {
  TRY {
  // bind to *any* bank service
  bank *b = bank::_bind (IT_X);
  // create a new account
  checkingAccount *cA =
    b->newCheckingAccount ("Grady", 1000.00, IT_X);
  // lodge $10
  cA->makeLodgement (10.00, IT_X);
  // output the limit
  cout << "limit is " << cA->overdraftLimit (IT_X)
                      << endl;
}
```

Figure 14. Client application code using inheritance

and storage, event handling, naming and directory services, and transactions.

ORB vendors are also adding features that should help the acceptance of the ORB technology. For example, Orbix allows server programmers to indicate specific-purpose proxies for the services to clients, and provides hooks for transparent filtering and interpretation of invocations.

ORB technology can help compose application components in the same address space, in the same machine, and across different machines including cross-language and cross-operating system bindings.

Annrai O'Toole, IONA Technologies Ltd., 8-34 Percy Place, Ireland. Telephone: 353-1-6686-6522. In the U.S., 1-800-ORBIX4U. E-mail: info@iona.ie. Mr. O'Toole , a co-founder of IONA, is vice president for development at IONA Technologies. He is responsible for technical and business strategy. He has degrees in Engineering from Dublin University and Trinity College and an MSc in Computer Science from Trinity College.

Configuring the IBM 7318 Terminal Server

By Eddie Ho and David Phipps

Users can connect to a RISC System/6000 (RS/6000) either through multiport adapters or a LAN. The Serial Communications Network Server (7318) allows LAN-based workgroups to access system resources. This article provides configuration tips for the Model P10.

The IBM 7318 Model P10 is a terminal server and the Model S20 is a communications server. Both provide a way to attach ASCII devices—such as terminals, modems, or printers—to AIX 3.2 systems using Ethernet™ wiring.

The 7318 Model S20 communications server is a stand-alone TCP/IP node that provides networking capabilities, such as Simple Network Management Protocol (SNMP), Kerberos security functions, and other serial protocols such as Serial Line Internet Protocol (SLIP), Compressed SLIP (CSLIP), and Point-to-Point Protocol (PPP) for remote TCP/IP communications.

The 7318 Model P10 terminal server off-loads terminal protocol processing from the RS/6000™. It provides the RS/6000 with TTY devices that behave like standard I/O serial ports, such as sa0 and sa1. These ports are functionally equivalent and interface compatible with the TTYs created with a multiport adapter. A high-performance protocol that is transparent to users is used over the SPX/IPX transport protocol between the AIX device driver and the server to provide the TTY devices.

Using SMIT

The 7318 P10 configuration panels can be accessed using System Management Interface Tool (SMIT). If the `cns.p10.obj` package has been installed, you can fastpath from the AIX prompt using `smit ts7318_mnu` or make the following selections in SMIT (root user authority is required):

- Devices
- Communications
- Serial Communications Network Server (ComNetServer)

Figure 1 shows the structure of the configuration menus. The order of the selections in the main menu can be mapped to the sequence of steps to define all resources on the 7318. For example, a terminal server must be configured before adding any TTY or printer devices.

Figure 2 shows three hosts and three device nodes, how each is addressed, and the configuration file entries for the hosts. It also shows the network definition for the topology.

IPX Addressing

NetWare® IPX network addressing principles are different and often simpler than for IP. Each host on the network has a user-designated internal network number that is stored in the configuration file `/etc/netware/NPSConfig`. This internal network number must be unique among all hosts on the network.

The administrator assigns a number to each Ethernet segment in the network. Hosts, attached to one or more Ethernet segments, can perform routing between these segments. A configuration file in the host lists the external network numbers to which it is attached, and associates the

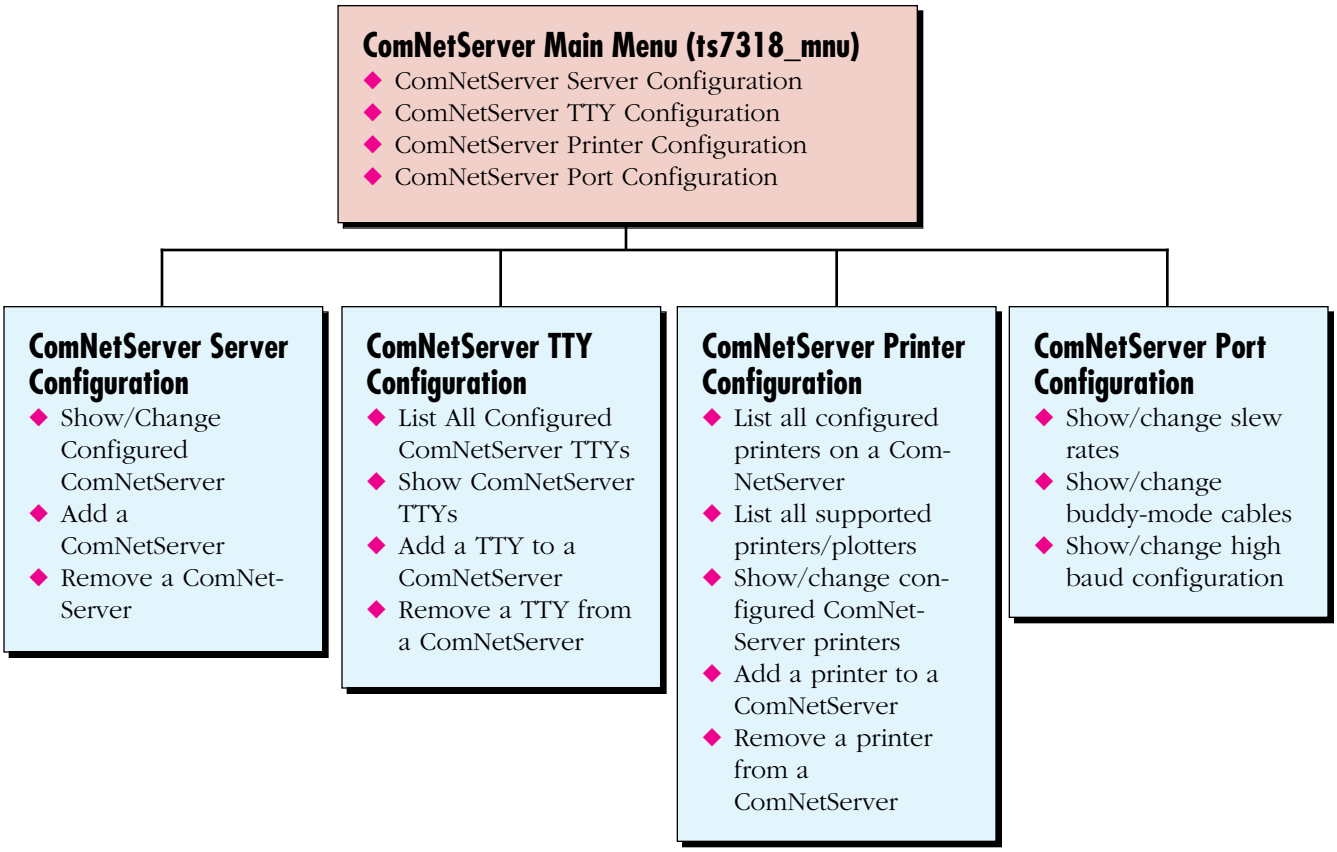


Figure 1. Configuration panels and navigation flows

numbers with the appropriate Ethernet adapter. The internal network numbers and the external Ethernet segment numbers must be unique. Both are logical network numbers—eight hexadecimal digits each.

A standard IPX address is represented by nnnnnnnn:ddddddddddd. The nnnnnnnn is the 8-digit internal network number and the ddddddddddd is the 12-digit Ethernet hardware address. Since hosts pretend to be a network by themselves, their Ethernet hardware address is not used as part of their node addressing. They are addressed by their internal network number with a fixed node address of 000000000001. All other devices on the network are addressed by the Ethernet segment number for the segment to which they are attached, followed by their Ethernet hardware address.

Configuring a 7318 Model P10

Figure 3 illustrates the two primary steps in configuring a terminal server. It includes two IPX nodes, the RS/6000, and a 7318-P10 connected by Ethernet. The 7318 has 16 IBM 3151 terminals attached to 16 serials ports; an IBM 4029 printer is attached to parallel port 1. The Ethernet interface in the RS/6000 is en0.

Step 1: Update the Configuration File

The configuration worksheet from the *7318 Serial Communication Server Guide and Reference* (SA23-2542) should be used to gather all networking information. This information can help resolve network address conflicts and can make configuration easier. It is also a roadmap for debugging networking problems.

From the worksheet, verify the parameter settings below and change them if necessary in the `/etc/netware/NPSConfig` file. Use your favorite editor to input the changes. Any change in this file requires a system reboot before proceeding to step 2. The following key parameters are those most frequently used:

- **spx = active**. This parameter enables the SPX/IPX protocol on this host. Change this value from `inactive` to `active`.

Sample IPX Network

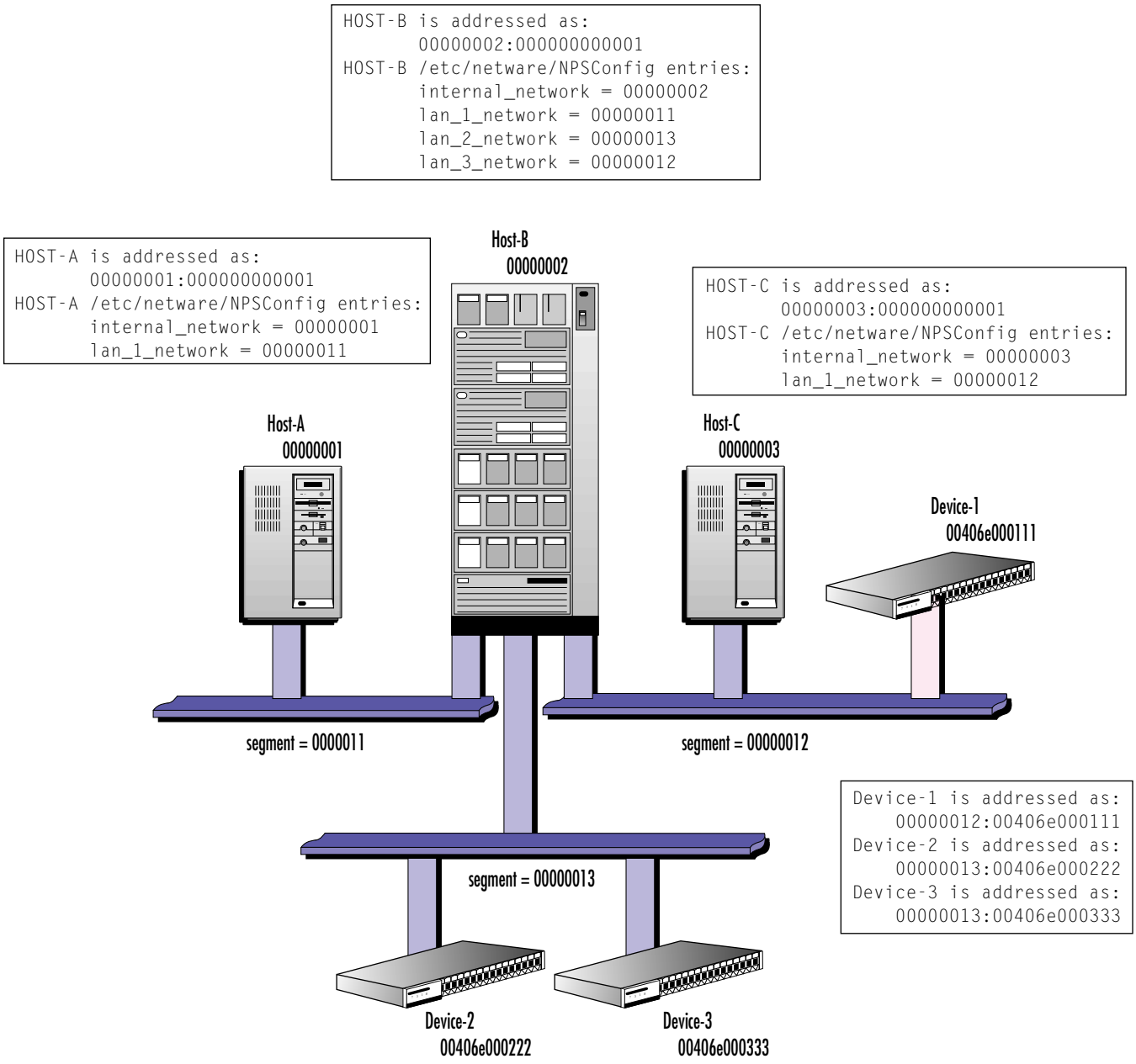


Figure 2. Sample IPX network with three hosts on three different segments

- **internal_network = 00000001**. This is a unique number for this host. Since no other hosts are in this example, the value will not conflict with hosts. However, it must be different from the LAN network number and the internal network number of all other hosts on the network.
- **lan_1_ppa = 0 and lan_1_if_name = "en"**. These two parameters contain the RS/6000 network interface value. Concatenate the values from `lan_x_if_name` and `lan_x_ppa` to obtain the Ethernet interface being used for this segment. For this example, these are `en` and `0`, yielding interface `en0`.

The next step is to ensure that this interface is enabled. Since IP and IPX can share the same Ethernet adapter, this interface is usually up if TCP/IP is already active. The `smit chinet`

Sample 7318 Topology

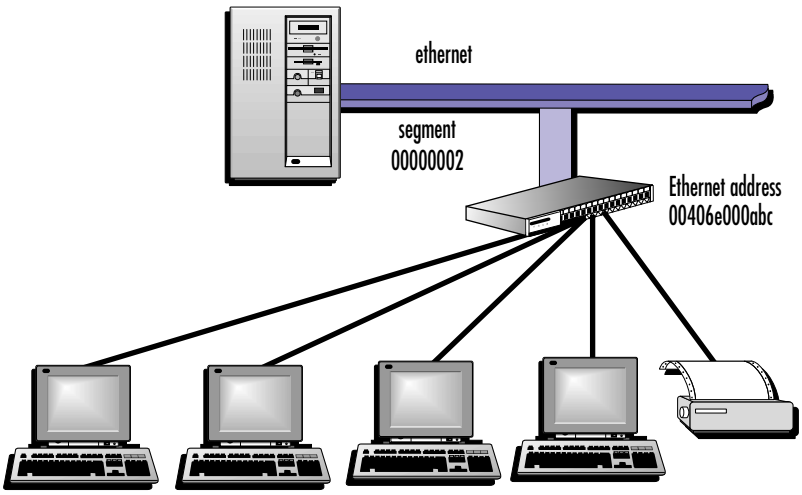


Figure 3. 7318 topology example

Parameter	Value
New ComNetServer Number	Contains the next available value.
ComNetServer Network Address	Contains the value set for `lan_1_network` in the `NPSConfig` file. This should be for the Ethernet segment on which the 7318 is attached.
ComNetServer Ethernet Address	The 12 hex digit Ethernet address. This address is located on the back of the 7318.

Figure 4. Values for panel parameters

```
                Add a ComNetServer

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                Entry Fields
* New ComNetServer Number       01
  Autogenerated TTY Devices     TTYs Only
  Interface Type                RS-232
* ComNetServer Network
  Address                       00000002
* ComNetServer
  Ethernet Address              00406e0002e8
  ComNetServer Bootfile         /usr/lib/cns/cns160

F1=Help      F2=Refresh    F3=Cancel    F4=List
F5=Reset     F6=Command    F7=Edit      F8=Image
F9=Shell     F10=Exit      Enter=Do
```

Figure 5. SMIT panel for adding a terminal server

fastpath will enable you to check this quickly, selecting the interface name you constructed above. Look in the Current `STATE` field for a value of `UP`. If the interface is not `UP`, then go ahead and start TCP/IP on this interface.

- **lan_1_network = 00000002**. This is a unique number for this host. This value defaults to 00000002. If IPX is running on the network, make sure that this value does not conflict with the internal network number of any other host, or the external network number of any of the Ethernet segments in the network.

Step 2: Use SMIT to Configure the Server and Its Attached Devices

Using the configuration worksheet, configure the 7318 unit. From the ComNet Server main menu, select the following:

ComNetServer Server Configuration
Add a ComNetServer

Use the values shown in Figure 4 to update the panel parameters shown in Figure 5.

This step configures the 7318 to AIX and automatically generates one RS-232 device session on each port of the 16 serial ports. If necessary, SPX/IPX will be started and the `CNSVIEW` daemon will be started or re-initialized with the new information.

The daemon monitors the network for the 7318 boot request and provides a command-line interface to perform administrative functions for the 7318. The 7318 can then boot from the system when it is powered up. This step takes about 2 minutes because of auto-TTY generations.

Configuring a Terminal

Configuring a terminal is usually not required for an ASCII terminal device. All 16 TTY devices were automatically created and configured to provide an AIX login prompt effective at the next system boot.

To enable them immediately, issue the `penable` command for each of these ports at the console (for example, `enable tty15`) after the TTY devices become available. The 7318 must finish booting before the configured devices are changed to the available state. You can check the state of each TTY device with a command such as `lsdev -C | grep tty15`.

Configuring a Parallel Printer

Printer definitions are not generated automatically—each port must be configured with a supported printer type. From the ComNet Server Printer Configuration menu in SMIT, select the "Add a Printer to a ComNetServer" option (shown in Figure 6) to define a printer device. Use the F4 key for a list of the printers supported by AIX. The `Printer ComNetServer Number` field must contain the appropriate number where the printer should be attached. The printer port number should be either p1 or p2. When completed, this step creates a special device file, such as `/dev/lpx` for the printer, and can be used by the printer spooling subsystem.

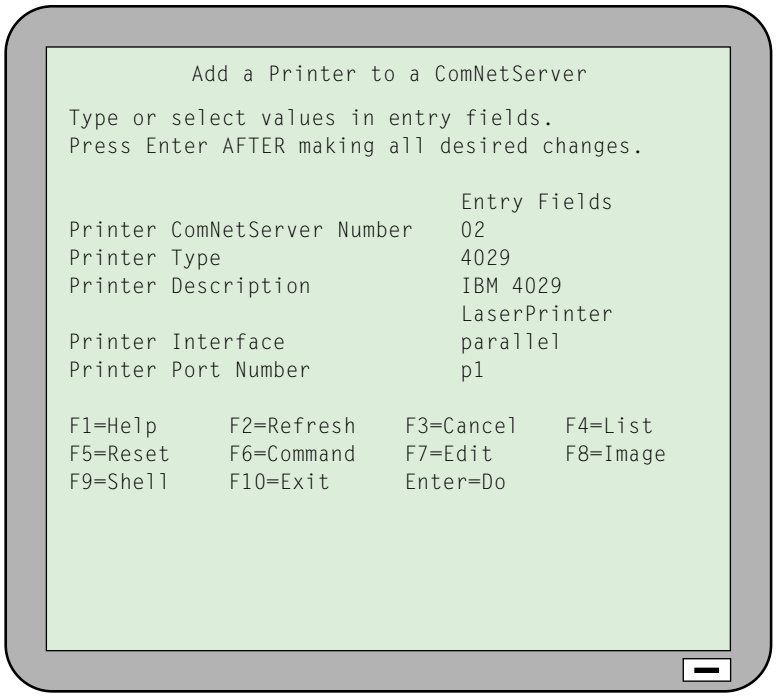


Figure 6. SMIT panel for adding a printer device to a terminal server

Eddie Ho, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Ho is a senior programmer in the AIX Executive Briefing Center. He has a BS in Computer Science from the University of Wisconsin and an MS in Computer Science from North Dakota State University.

David Phipps, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Internet: dlphipps@ausvm6.vnet.ibm.com. Mr. Phipps is a staff programmer in the AIX Communications area. He has a BS in Applied Mathematics, Engineering, and Physics and an MS in Computer Science from the University of Wisconsin.

Providing Highly Available NFS Services with HACMP

By Thomas Casey and Robert Metcalf

IBM's HACMP software is a platform for highly available services. This article will help current HANFS users determine if HACMP satisfies their requirements. It describes how each product provides highly available NFS services, including the advantages and disadvantages of each approach. It also provides a general "how to" for using HACMP to build a highly available NFS server, including special considerations and caveats.

Thomas Casey

Robert Metcalf

The Network File System (NFS) has gained universal acceptance as a general-purpose "shared" filesystem for client/server computing. AIX users have two options for making NFS services highly available: High Availability for Network File System (HANFS) and High Availability Cluster Multiprocessing (HACMP).

Each product has distinct strengths. HANFS provides a reliable NFS server capability by enabling a backup processor to recover current NFS activity if the primary NFS server fails. HACMP provides a general infrastructure on which to build highly available environments for mission-critical data and applications.

The main advantage of HANFS is that it uses AIX extensions to the standard NFS functionality, enabling it to handle duplicate requests correctly and restore lock state during NFS server fallover and reintegration. The main advantage of HACMP is its flexibility and scope. HACMP supports more fallover configurations, processors, disks, and networks than HANFS.

Currently, HANFS has an uncertain future. Although it continues to be included in AIX 3.2.5, IBM has not made it available on AIX 4.1. This gives users the choice of staying with the current version of HANFS and accepting the current AIX, hardware, and configuration support, or finding an alternative way of providing highly available network filesystem services.

HANFS

HANFS, an option of AIX 3.2, is designed to make NFS services highly available. HANFS supports the NFS protocol, can be used by NFS clients without modification, and provides reliable filesystem services by recovering from disk and server failures.

Recovering from Disk Failures

HANFS handles disk failure by using the Logical Volume Manager (LVM) mirroring facility to mirror data across different physical volumes. All copies of the data are on disks controlled by the same file server, which eliminates the overhead of ensuring consistency and coherence between the two servers. If a disk fails, the LVM redirects disk requests to a mirrored copy, making the disk failure transparent to users.

Recovering from Server Failures

HANFS handles server failures by connecting dual-ported disks to a pair of RISC System/6000 processors. One processor in the pair is designated as the server, the other as the backup. The server maintains enough information on the shared disks so that the current state at the time of failure can be reconstructed.

The two processors periodically exchange heartbeat messages to monitor the state of the other processor in the server pair. If the server

fails, the backup processor takes over the shared volume groups and uses the information stored there to reconstruct the lost state. The backup then impersonates the failed server and normal operations continue. NFS clients on the network are oblivious to the failure and access the filesystem at the same address.

HANFS Functional Overview

HANFS supports two processors arranged in one of two possible configurations:

- One processor is the server and the other processor is strictly a backup.
- Each processor is both a server and a backup.

When a client mounts a directory from the HANFS server, the client's name is registered in a special file on both the local and remote processors. If the server fails, the backup detects the failure, takes over for the server, and reestablishes NFS services. Using its client list, the backup contacts all clients, which then re-request any outstanding locks they held on the server. Later, when the primary server is operational, this procedure is reversed and the server once again provides the NFS services.

The HANFS software consists of three daemons:

- **HANFS** (`hanfsd`). The `hanfsd` daemon coordinates all actions between the server pair. It starts the `hacfgd` and `hapngd` daemons, and tells the `hacfgd` which operations to perform and when to perform them.
- **HANFS configuration** (`hacfgd`). The `hacfgd` daemon has three major functions: configuring the server pair, coordinating the fallover to the backup when the server fails, and reintegrating the server when it returns.
- **HANFS ping** (`hapngd`). The `hapngd` daemon issues ICMP ECHO requests to determine when its partner processor joins or fails, and reports changes in status to the `hanfsd` daemon.

Configuring the Server Pair

The `hacfgd` daemon performs the following tasks to configure the server pair:

1. Mounts the filesystems
2. Stops the NFS `nfsd`, `rpc.mountd`, and `rpc.lockd` daemons
3. Exports the filesystems listed in the `/etc/exports.hanfs` file
4. Sets up the primary adapter
5. Starts the NFS `rpc.mountd`, `rpc.lockd`, and `nfsd` daemons
6. Refreshes the `rpc.statd` daemon on the backup processor

JFS Logging and the NFS Duplicate Cache

A duplicate cache allows NFS to reject duplicate requests that, if executed, might have harmful side effects. For example, a duplicate delete of a file would fail but a duplicate read would succeed. The `nfs_dupget` and `nfs_dupsave` commands are AIX extensions to NFS that copy the duplicate cache into user memory and restore duplicate cache entries from user memory. For each entry in the NFS duplicate cache, the NFS daemon creates a Journaled File System (JFS) log entry so that the NFS duplicate cache can be rebuilt during the log redo portion of the filesystem consistency check. Since the cache is non-volatile, the backup processor can restore the cache after an NFS server has crashed.

The main advantage of HACMP is its flexibility and scope.

rpc.statd Extensions

AIX provides extensions to the NFS `rpc.statd` daemon that instruct it to inform the backup node when new clients request NFS services. This remote node then creates the `/etc/sm/<hostname>` files that shadow the server's `/etc/sm/<hostname>` files so that client lock requests can be tracked and reclaimed after fallover.

Fallover

After the server fails, the `hacfgd` daemon performs the following steps to acquire the primary server's NFS mount points:

1. Mounts the filesystems (includes `fsck` and log redo)
2. Refreshes `rpc.statd` on the remote processor
3. Stops the NFS `nfsd`, `rpc.mountd`, and `rpc.lockd` daemons
4. Exports the filesystems listed in the `/etc/exports.hanfs` file
5. Takes over the server's primary adapter address
6. Starts the NFS `nfsd`, `rpc.mountd`, and `rpc.lockd` daemons

Restoring NFS Duplicate Cache

The `fsck` and log redo performed on the filesystem rebuild the NFS duplicate cache on the backup node so that duplicate requests are handled correctly.

Restoring Lock State

Since the backup has a record of all clients that held locks on the server (the `rpc.statd` extensions), it informs the clients to resubmit their lock requests so that the lock state can be rebuilt. The NFS `rpc.lockd` daemon performs this operation during its initialization phase.

Reintegration

When the server is restarted after a failure, the backup performs the following operations:

1. Stops the NFS `nfsd`, `rpc.mountd`, and `rpc.lockd` daemons
2. Releases the filesystems
3. Releases the primary adapter of the server
4. Starts the NFS `nfsd` daemon
5. Refreshes the NFS `rpc.statd` daemon on the server
6. Starts the NFS `rpc.mountd` and `rpc.lockd` daemons
7. Copies the NFS duplicate cache to the server

When this completes, the primary server then performs the configuration steps described in the configuration section above.

Advantages and Disadvantages of HANFS

HANFS takes advantage of AIX extensions to the standard NFS functionality so that it can handle duplicate requests correctly and restore lock state during NFS server fallover and reintegration. Although NFS supposedly does not maintain any state information, most real-world implementations maintain a small amount of state information in a duplicate cache.

HANFS uses the AIX-supplied extensions to record duplicate request and outstanding lock information on the shared disk so that state information is not dependent on a single processor. If a failure occurs, the backup can read this information from the shared disk and reconstruct the duplicate cache and restore the current locks. In this way, failure and recovery are completely

Building a Highly Available NFS Server with HACMP

Building a highly available NFS server with HACMP requires almost as much time designing the cluster in front of a whiteboard as it does implementing the cluster in front of a console. As you plan the cluster, determine whether you need a server-to-server or server-to-client configuration and devise a strategy for fallover and reintegration. Here are some issues to consider and some pitfalls to avoid.

- If you choose a server-to-server implementation, make sure that your takeover node has the capacity to perform its original duties and the additional work of the failed node. Also, on a server-to-server cluster, devise a plan to deal with the increased fallover time.
- During fallover, the takeover node resets the disks or controllers of the shared disks and varies on the volume group. The filesystem is then mounted locally and NFS-exported to clients. In a server-to-server configuration, the reset is preceded by the `clnfskill` command followed by a umount on the NFS-mounted filesystem from the failed host. You can customize this default behavior through script changes (not recommended) and event management (recommended).
- You can use pre- and post-event processing to reduce fallover time in server-to-server configurations. Use pre-events to batch the umounts rather than run them serially, which significantly reduces umount time. Use post-events to handle all NFS processing on the takeover node and to granularize NFS handling beyond default HACMP behavior.
- For fallover to appear seamless to the clients, major numbers must be the same for NFS volume groups on all server nodes.
- By default, HACMP runs the `exportfs -i` command during fallover, ignoring the `/etc/exports` file. You may need to modify this processing at your site.
- Be sure you consider and plan for reintegration of a failed node. If you choose to have a node reintegrate, your clients will see an interruption in service while the disk, volume group, and filesystem are dropped by the takeover node and reacquired by the original node.
- HACMP provides the `clnfskill` utility to kill processes with files open in an NFS-mounted filesystem. Read the man page for this important command to be sure you understand how and when to use it.

transparent to applications running on the file server's clients.

HANFS also has some disadvantages:

- HANFS is not included in AIX 4.1.
- An HANFS configuration is limited to two processors. If both processors fail, NFS services cannot be restored to clients.
- HANFS is limited to SCSI support only. Other disk technologies, such as IBM 9333 serial link disks, are not supported.
- Since HANFS does not support AIX 4.1, you cannot use symmetric multiprocessors with HANFS.
- HANFS is limited to NFS functionality only.
- HANFS does not support server-to-server (cross-mount) NFS functionality; it supports server-client only.

HACMP

HACMP provides high availability and cooperative computing services for clusters of two to eight RISC System/6000s. HACMP software combined with industry-standard hardware reduces downtime by quickly restoring services when a critical system, component, or application fails. A cooperative computing system enables distributed applications to take advantage of the increased computing power available in the cluster.

The key facet of a highly available system is its ability to detect and respond to changes that could impair essential services. HACMP enables a cluster to continue to provide critical data and applications services even though a key system component, such as a network adapter, is no longer available. When a component becomes unavailable, HACMP detects the loss and shifts that component's workload to another component in the cluster.

Although the switchover is not instantaneous, services are restored rapidly, usually within one to five minutes. Since high availability is not fault tolerant, many sites are willing to absorb a small amount of downtime rather than pay the much higher cost for fault tolerance.

A highly available cluster is built by eliminating single points of failure in the cluster. A single point of failure exists when a critical function is provided by only one component. If that component fails, the cluster has no other way to provide that function and its services become unavailable. For example, if all the data for an application resides on a single non-mirrored disk and that disk fails, the disk is a single point of failure for the entire cluster. Clients cannot access that application until access to the data on the disk is restored.

HACMP provides recovery options for the following system components:

- Processors
- Networks and network adapters
- Disk and disk adapters
- Applications

The HACMP Cluster Manager

The cluster manager agent is the central control mechanism for providing highly available services. It runs as a daemon process on each processor, or node, in an HACMP cluster. The cluster manager monitors local hardware and software subsystems, tracks the availability of other nodes in the cluster, and coordinates the takeover and release of cluster resources in response to changes in the cluster topology, known as cluster events. A cluster event can be triggered by a change affecting a network adapter, network, or node.

HACMP can deal with various types of failures including network adapter, network, node (including application failure), and disk and disk adapter failure (using LVM mirroring).

HACMP provides high availability and cooperative computing services for clusters of two to eight RISC System/6000s.

Recovering from Network Adapter Failures

A service adapter is the primary connection between a node and a network. The cluster manager monitors network adapters by sending keep-alive packets every half second over the service adapters. The cluster manager uses the presence or absence of keep-alive activity over a service adapter as a sign of the adapter's health. If there is no keep-alive activity over an adapter for five seconds, the cluster manager instructs a standby adapter to take over the IP address of the service adapter.

Once detected, swapping adapters takes about three seconds to complete. Client applications do not lose their connection. A slight delay may be noticeable, but often there is no discernible effect beyond normal workload variations. Therefore, a network adapter failure is detected and fully recovered in less than 10 seconds.

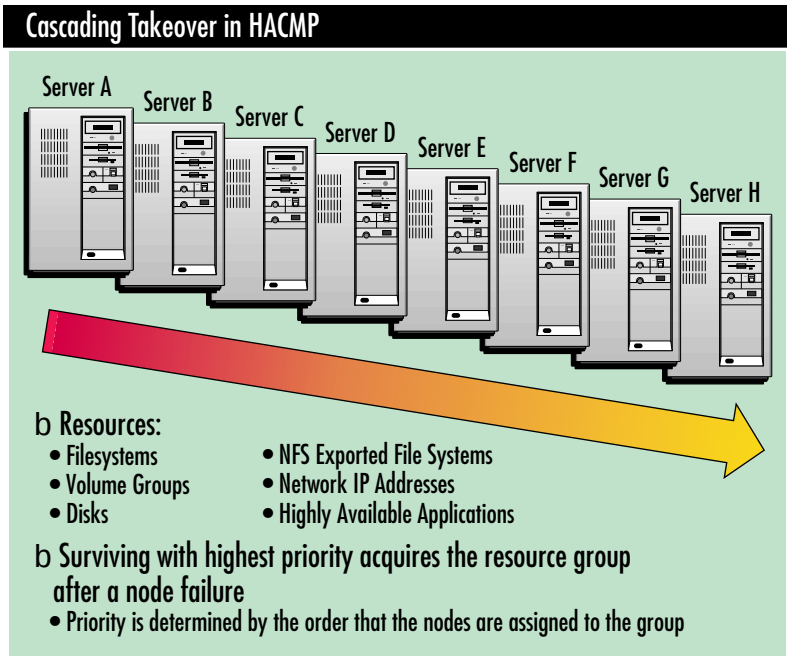


Figure 1. Cascading takeover available in HACMP

Recovering from Network Failures

HACMP uses multiple communications links to eliminate a network as a single point of failure. The cluster manager detects when a network fails and when it later returns to service. Detecting a local network event takes three seconds; detecting a remote network event takes six seconds. The cluster manager sends a mail message to the system console when a network event occurs, but takes no further action since the appropriate response depends on the specific network configuration. Using HACMP's event customization facility, you can tailor the fallover processing to your site.

Recovering from Node Failures

An HACMP node owns a set of resources: disks, volume groups, filesystems, networks, network addresses, and applications. When a node fails or leaves the cluster, some or all of its resources are distributed among the surviving nodes.

The cluster manager monitors the availability of the other nodes in the cluster by exchanging heartbeats with its neighboring nodes. If a node stops sending heartbeats, the cluster managers on the surviving nodes in the cluster take the necessary actions to get the critical applications up and running and to ensure that data has not been corrupted or lost. The available cluster managers take over the network interface, the volume groups, or disk drives, and then restart the applications.

As part of the takeover, the cluster managers delete and re-create their routes, and refresh the Address Resolution Protocol (ARP) cache of any clients. This allows clients to connect to the backup node using the same address originally used to connect to the primary node.

Any client transaction in mid-flight during a node failure must be resubmitted. This does not cause data corruption since the in-flight transaction has not yet been committed.

Clients experience a brief interruption in service during node failure—typically from one to five minutes—and then continue processing. The takeover time varies depending on the amount of resources that need to be shifted to the takeover node and the amount of application recovery processing required.

Recovering from Disk and Disk Adapter Failures

Like HANFS, HACMP does not directly recover disk and disk adapter failures. These failures are handled within AIX by mirroring across different physical volumes on disks and enclosures, and by internal data redundancy and redundant adapters on disk arrays. In addition, HACMP provides a System Management Interface Tool (SMIT) interface to the AIX Error Notification facility. This allows a system administrator to detect a failure not specifically monitored by HACMP—such as a disk adapter failure—and to program a response to the failure.

Advantages and Disadvantages of HACMP

Using HACMP to provide highly available NFS services has many advantages:

- HACMP is a mature product with a large installed base (over 3,000 licenses). The product's functionality and usability have improved significantly since its initial release in 1992. Future releases will continue to address needs of the RISC System/6000 community.
- HACMP provides extensive processor, disk subsystem, and network support. HACMP supports RISC System/6000 uniprocessor and Symmetric Multiprocessor (SMP) system units. HACMP also supports the POWERparallel SP2 machine. HACMP supports SCSI, SCSI-2, 9333 serial link disk subsystems, as well as 3514, 7135, and 7137 disk arrays. HACMP supports Ethernet, Token Ring, Serial Line Internet Protocol (SLIP), Serial Optical Channel Converter

(SOCC), Fiber-optic Distributed Data Interchange (FDDI), and Fiber Channel Standard (FCS) networks. New processors, disk devices, and communication devices are regularly tested and approved for HACMP.

- An HACMP cluster providing highly available NFS services is not limited to NFS. Other highly available services (such as local disk, concurrent access, and communication) can be included in an HACMP cluster designed primarily for NFS services.
- HACMP supports numerous cluster configurations: traditional *hot standby* configurations where one or more standby machines back up one or more servers; *mutual takeover* configurations, which partition the workload across the cluster; and *concurrent access* configurations where multiple machines simultaneously access a shared database.
- HACMP offers cascading takeover. A single NFS server in a cluster can be backed up by all the other servers in the cluster, for a total of eight nodes. If the NFS-serving node fails, its role is taken over by another node. If that node fails, the NFS-server role cascades to another node, and so on, as shown in Figure 1.
- HACMP's concurrent access feature allows two to eight processors to simultaneously access a database residing on a shared external disk. Using concurrent access, a cluster can provide near-continuous availability that rivals fault tolerance, but at a much lower cost. Additionally, concurrent access provides higher performance, eases application development, and allows horizontal growth.
- The HACMP lock manager is a cooperative computing service that allows distributed applications to serialize access to concurrent resources with locks. The lock manager interface consists of a daemon that maintains a common cluster-wide database of resources against which locks can be taken, and an Application Programming Interface (API) that clients use to request locks.
- The cluster information facilities provided with HACMP enable a developer to monitor the cluster state and to write cluster-aware applications. These facilities include the `clsmuxpd` and `clinfo` daemons. The `clsmuxpd` daemon maintains a network Management Information Base (MIB) accessible to clients via the standard Simple Network Management Protocol (SNMP) interface for network management applications.

  Cluster-aware applications that do not want to use the complex SNMP interface can obtain cluster information using the simpler `clinfo` interface. The `clinfo` interface consists of a daemon that updates cluster topology information in a shared memory with information retrieved from the `clsmuxpd` daemon, and an API that the client applications can use to retrieve this information.

Server-to-Server NFS Cross-mounting

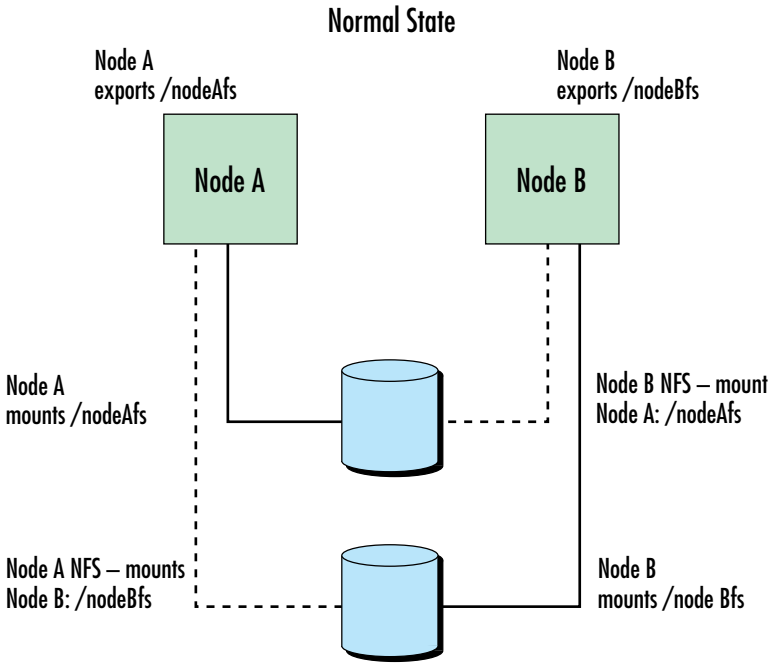


Figure 2. Normal state of server-to-server NFS cross-mounting

- HACMP supports server-to-server (cross-mount) functionality, as well as server-to-client. In other words, Node A can mount `/nodeAfs` locally, and export it to Node B. Node A can then NFS-mount a filesystem (`/nodeBfs`) offered by Node B. Node B can mount `/nodeBfs` locally, and export it to Node A. Node B can then NFS-mount the filesystem (`/nodeAfs`) offered by Node A, as shown in Figure 2. If either node fails, the other can then locally mount and re-export the failed filesystem to other NFS clients, as noted in Figure 3.

Server-to-Server NFS Cross-mounting

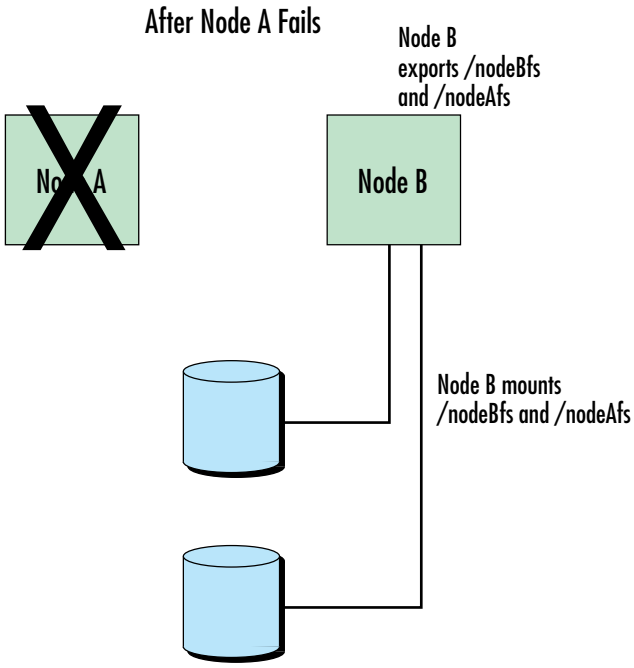


Figure 3. After a node fails in server-to-server NFS cross-mounting

There are also some drawbacks to using HACMP for highly available NFS services:

- Unlike HANFS, HACMP does not take advantage of the NFS extensions provided by AIX. If a failure occurs, applications using file locking need to ensure that locking information gets to the takeover node.

  In a server-to-server configuration where the node doing the NFS mount is using file locking, HACMP may need to remove the `/etc/sm.bak/$host` files before attempting fallover. Use the HACMP `cl_deactivate_nfs` utility to remove these files, followed by a `stopsrc/startsrc` of the `rpc.statd` and `rpc.lock` daemons.

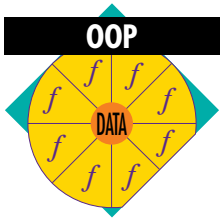
- In server-to-server clusters where filesystems are cross-mounted for one server to another, fallover time is increased. Using the default HACMP scripts, the `clnfskill` utility is run on the NFS-mounted filesystem, followed by a umount. The umount takes 45 to 60 seconds per filesystem as it waits for the RPC timeout from the failed node. If the default HACMP NFS processing is used and there are many cross-mounted filesystems, the fallover time increases significantly.

Thomas Casey, CLAM Associates, Inc., 101 Main Street, Cambridge, MA 02142. Internet: tom@clam.com. Mr. Casey is the manager of CLAM's technical writing group. He has a BS from Trinity College in Hartford, Connecticut and an MS from Emerson College in Boston.

Robert Metcalf, CLAM Associates, Inc., 101 Main Street, Cambridge, MA 02142. Internet: bobmet@clam.com. Mr. Metcalf is a senior member of CLAM's technical support staff. He has a BS from Suffolk University and an MS from Simmons College, both in Boston.

Building Object-Oriented Frameworks

Part 2



By Deborah Adair

By using frameworks, Taligent®, Inc., an independent joint venture of Apple® Computer, IBM, and Hewlett-Packard®, is realizing the full promise of object technology. A framework defines the behavior of a collection of objects, providing an innovative way to reuse software designs and code. Part 1 (see the February 1995 issue) described different types of frameworks, how they are used, and addressed organizational concerns that impact framework development. Part 2 outlines a process for identifying and designing frameworks.

Developing a framework differs from developing a stand-alone application. A successful framework solves problems that appear different from the problem that justified its creation.

Developing Frameworks

The first step in developing a framework is to analyze your problem domain and identify the frameworks you need. Steps for each framework are to identify the primary abstractions, design how clients[1] interact with the framework, and implement, test, and refine the design.

To illustrate these steps, this section describes a small program and one framework that could be used to build it. The program is a data-visualization tool called StockBrowser, which allows users to browse stock histories and display the information graphically. This program enables users to select different stocks and view the price trading volume graphically. The stock information can be displayed as daily, weekly, or monthly values, showing users information such as a stock price on a particular day, or which month had the heaviest volume of trading.

Analyzing the Problem Domain

Once you have identified the problem domain, divide the problem into a collection of frameworks that can be used to build a solution. If the problem maps to a process, describe the process from the user's perspective. For each framework, identify the process it models. Once you have outlined the process, you can identify the necessary abstractions.

Identifying Frameworks

Frameworks can be thought of as abstractions of possible solutions to problems. You can often identify opportunities for new frameworks by analyzing existing solutions.

To determine what frameworks you need, consider families of applications rather than individual programs.

Deborah Adair

[1] This article is written from the viewpoint of framework developers. We use the term "client" to refer to developers who might use a framework that you developed.

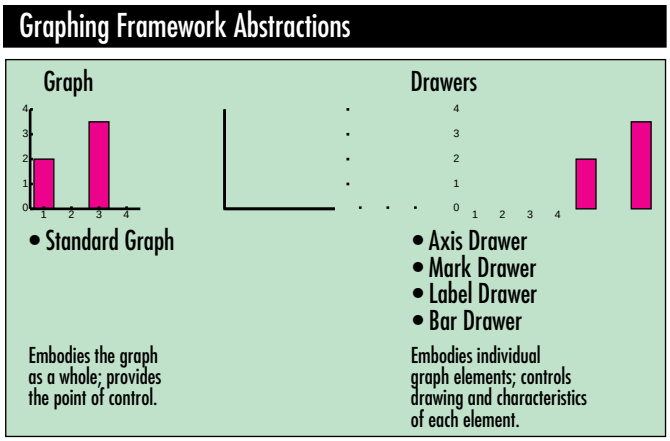


Figure 1. Graphing framework abstractions

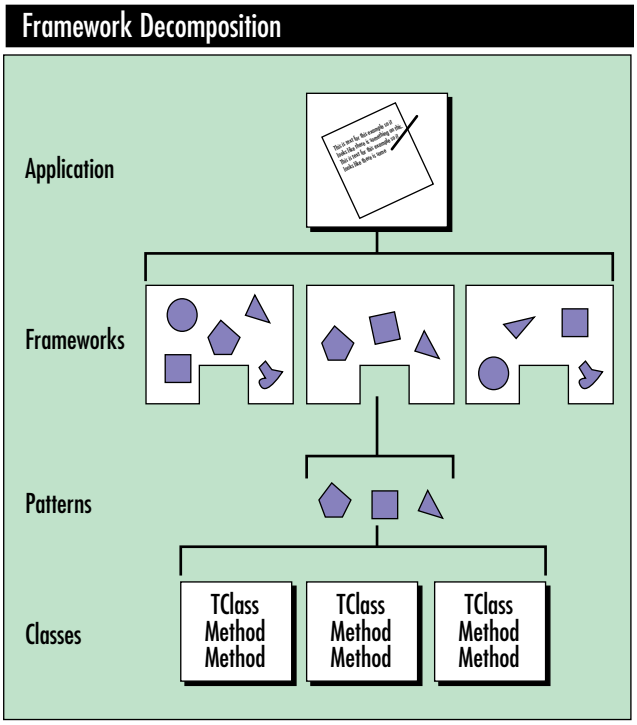


Figure 2. Framework decomposition

- Look for software solutions that you build repeatedly, particularly in key business areas.
- Identify what the solutions have in common and what is unique to each program.

The common pieces—the parts that are constant across programs—become the foundation for your frameworks. Factor these pieces into small, focused frameworks.

When a suite of applications solves similar problems, there is often an opportunity for developing a framework. Potential frameworks can be found in real-world models, processes performed by end users, and source code for current software solutions.

One framework that could be used to build the StockBrowser program is a 2-D graphing framework. The graphing framework would provide support for drawing and labeling the axes and for plotting the data. This type of framework could be used in a wide variety of data-visualization applications—from executive information systems to scientific visualization programs.

Identifying Abstractions

Identify the abstractions your clients need to describe their problems and then provide the logic for producing a valid solution with those abstractions.

The easiest way to identify the abstractions is by using a bottom-up approach—start by examining existing solutions. Analyze the data structures and algorithms, then organize the abstractions. Always identify the objects before you map out the class hierarchy and dependencies.

If you are familiar with the problem domain, you can draw from your past experience and former designs to identify abstractions and begin designing your framework. If you are not an expert in the problem domain, or have not developed any applications for it, examine applications written by others and consider writing an application in the domain. Then factor out the common pieces and identify the abstractions.

There are two primary abstractions for the graphing frameworks: the graph itself and drawers for the graph. Separating the drawing information from the graph enables this simple framework to be extended. New graph elements can be added by providing new types of drawers. Figure 1 shows a graphing framework abstraction.

Designing the Framework

If the framework models a process, determine which steps in the process are performed by the framework, and which steps the client performs. As you design how the framework will work, you might also discover that you can break down the framework into a collection of recurring design patterns, similar to the way you decompose the initial problem into a set of frameworks, as shown in Figure 2.

Each design pattern is a micro-architecture for a recurring element. Patterns can represent generic software elements or elements that are particular to a problem domain. When you design a

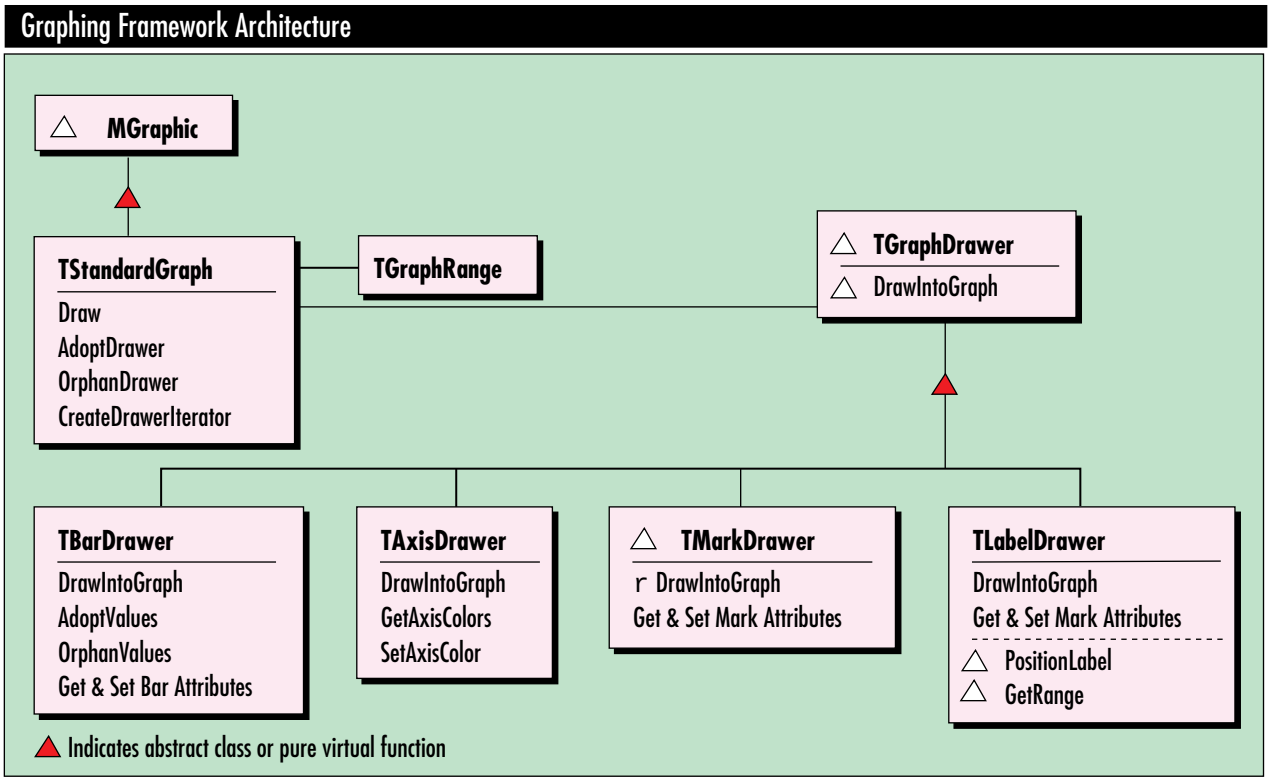


Figure 3. Graphing framework architecture

framework, look for recurring patterns that can be applied to other problems.

The graphing framework supports a simple process—drawing a 2-D graph. The client provides the data for the graph and tells the graph to draw, which then tells each of its elements to draw. The final appearance of the graph is determined by the data provided by the client and the characteristics of the drawers.

On one level, the graphing framework could be described as a manager-driven framework—where the graph functions as the manager. Most framework actions are triggered by telling the graph to draw.

Designing Client-Framework Interactions

In your framework design, focus on how the client interacts with the framework—what classes and member functions the client uses. Look for ways to reduce the amount of code that clients must write by doing the following:

- Provide concrete implementations that can be used directly.
- Minimize the number of classes that must be derived.
- Minimize the number of member functions that must be overridden.

Another consideration is how to support customization. Customization is a two-edged sword—you want flexibility, but you also want to maintain the focus of the framework and minimize the complexity for the client. A completely flexible framework will be difficult for your clients to learn and difficult for you to support.

One approach is to build a very flexible, general framework from which you derive additional frameworks for narrower problem domains. These additional frameworks provide the default behavior and built-in functionality, while the general framework provides the flexibility.

As you design the framework, consider how the design can help communicate the use of the framework. Class names, function names, and how you use pure virtual functions can all provide cues for using the framework.

You also need to determine how the framework classes and member functions interact with client code, such as what objects are created when the client calls framework functions, and when client overrides are called by the framework.

As with most frameworks, the graphing framework provides both features that the client can

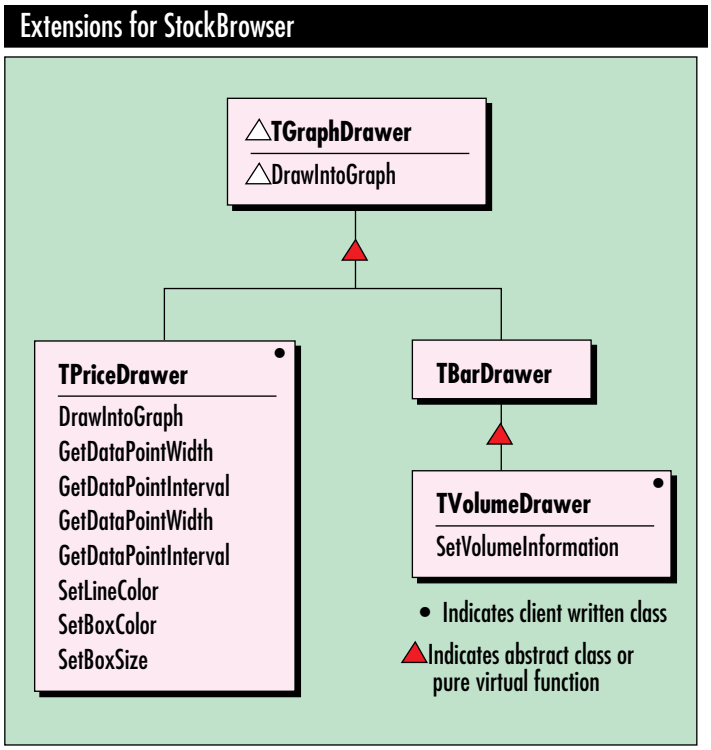


Figure 4. StockBrowser extensions

```
TGraphRange xAxisRange (0, 53);
TGraphRange yAxisRange (0, 100);
TGPoint axisLengths (530, 100);
TStandardGraph graph(xAxisRange,
     yAxisRange, axisLengths);
.
.
GraphValue dataPointWidth = 6;
GraphValue dataPointInterval = 1;
TPriceDrawer* priceDrawer =
    new TPriceDrawer(dataPointWidth,
    dataPointInterval, *data);
priceDrawer->SetLineColor(
    TRGBColor(.75, .2, .75));
priceDrawer->SetBoxColor(
    TRGBColor(0, 0, 0));
graph.AdoptDrawer(priceDrawer);
.
.
graph.Draw(*port);
```

Figure 5. Code for displaying stock data

use directly and an underlying structure that can be extended by deriving new classes. Even this simple framework has both data-driven characteristics and architecture-driven characteristics.

To use the framework, a client constructs the necessary drawers and a standard graph. When the client calls `TStandardGraph::Draw`, then `DrawIntoGraph` is called for each drawer, and the graph and all of its elements are drawn. To change the appearance of the graph, clients can call functions such as `TAxisDrawer::SetAxisColor`.

To extend the framework, clients can derive new drawers from `TGraphDrawer` or one of its subclasses. It is also possible to derive new graph types from `TStandardGraph`—for example, a new type of graph that supports buffered drawing (see Figure 3).

Figure 4 shows how two new drawers were added to the StockBrowser program: `TPriceDrawer` and `TVolumeDrawer`.

`TPriceDrawer` charts stock prices, displaying high, low, and close prices for a particular stock. This class is derived directly from `TGraphDrawer` and from `MStockHolder`, which holds a collection of daily stock data. `TPriceDrawer` overrides `TGraphDrawer::DrawIntoGraph`, and provides functions for manipulating the data points and attributes.

`TVolumeDrawer` charts stock trading volumes in a bar chart. This class is derived from `TBarDrawer`, a generic 2-D bar-chart element, and `MStockHolder`. `TVolumeDrawer` adds the function `SetVolumeInformation` to get the stock volume information to display.

To display the stock data, StockBrowser creates drawers for the graph elements, specifies attributes for the elements, and calls `TStandardGraph::Draw`. The graphing framework does the rest. Figure 5 shows the code for displaying stock data.

Refining the Framework

Building a framework is an iterative process. Beginning with your initial design, you should work with your clients to determine how the framework can be improved—implement features, test them, and verify them with your clients. During this process, you will reanalyze the problem domain and refine your design based on testing, client feedback, and your own insights.

As your framework matures, you will probably find more features to add, and possibly identify opportunities for entirely new frameworks or frameworks that support a particular subset of the problem domain.

Look for additional ways to make your clients' tasks easier. In some cases, it makes sense to provide special tools with your frameworks. Code

generators, CASE tools, and Graphical User Interface (GUI) builders can make programming with frameworks easier, just as they do for traditional software development.

Do not let your frameworks become too large; keep looking for ways to break them down into small, focused frameworks. If they are designed to interoperate, small frameworks are more flexible and can be reused more often.

The nature of an iterative process makes it difficult to determine when a framework is finished. A simple framework requires fewer iterations than more complex frameworks—another advantage of developing smaller, focused frameworks. Until you actually release the framework, you will not gain any of the benefits.

Although prototyping is not unique to framework development, it is very useful. A common approach is to implement a framework that applies to a specific subset of a larger problem domain, and then rework it to support more general cases. The ET++ SwapsManager project is one example of this approach.

After developing a general framework that provides a strong architectural base, you can derive additional frameworks that apply to particular problem sets. The overall framework provides generalized components and constraints to which the derived frameworks conform. Derived frameworks introduce additional components and constraints that support more specific solutions.

Derived frameworks are another method of providing default behavior for your clients. If your framework consists of several abstract classes, you might want to create one or more derived frameworks that provide concrete implementations and additional built-in functionality.

The document framework architecture in the Taligent system supports the creation of most components and documents that will be seen by end users. This set of frameworks was designed to be flexible and allow the construction of a wide range of programs. Derived frameworks support a narrower set of applications. For example, a graphical editing framework simplifies creating programs that manipulate graphical data. Many standard graphics editing functions such as move, scale, rotate, and group are built into this derived framework.

Documenting the Framework

Code comments and documentation are a necessary part of any programming project, but they are especially important if you develop frameworks. Other developers must understand your framework or they will not use it.

Learning how to use another developer's framework can be difficult. As a framework developer, you must provide enough information for your clients to understand how to use the framework for producing the solution they want.

Make it clear what classes can be used directly, what classes must be instantiated, and what classes must be overridden. Clients are interested in solving particular problems—not the details of the framework implementation. It is best to provide a variety of documentation including sample programs, diagrams of the framework architecture, descriptions of the framework, and descriptions of how to use the framework.

At a minimum, provide sample programs. In developing and testing your framework, you have to develop applications that use your framework; often these can provide a foundation set of samples. Try to provide a variety of samples that demonstrate how to use the framework in different contexts—a well-rounded set of sample programs is invaluable to clients learning your framework. Consider designing the sample programs so that they show a progression of the architectural features of the framework.

After developing a general framework that provides a strong architectural base, you can derive additional frameworks that apply to particular problem sets.

Managing Change

Frameworks evolve, especially as your understanding of the problem domain and number of clients grows. However, once you release a framework for client use, you should limit the changes—a constantly changing framework is difficult, if not impossible, to use. As a rule, fix bugs immediately, add new features occasionally, and change interfaces as infrequently as possible.

During framework development, changes happen much more frequently. Once clients have been using a framework, you might still need to make changes that impact their work.

When you update a framework, minimize the impact on your clients. It is better to add new classes instead of changing the existing class hierarchy, or to add new functions instead of changing or removing existing functions.

Give your clients advance notice of framework updates and allow time for them to adapt to the changes. One approach is to add the new and changed classes in an interim release and flag the old ones with obsolete warnings. This warns your clients before their code is broken that they are using classes that are going to change.

Maximizing Framework Benefits

For you and your clients to get the maximum benefit from a framework, the framework should have the following characteristics:

- Provide as much built-in functionality as possible
- Minimize the potential for client errors
- Follow standard design and coding guidelines
- Be extensible
- Be portable

Guidelines for Developing Frameworks

Here are some simple guidelines for developing frameworks:

- Derive frameworks from existing problems and solutions.
- Develop small, focused frameworks.
- Build frameworks using an iterative process driven by client participation and prototyping.
- Treat frameworks as products—provide documentation and support, and plan for distribution and maintenance.

As you develop a framework, you make many small iterations on the design and initial implementation. As the framework matures and your clients begin to rely on it, the changes become less frequent and you move into a much larger maintenance cycle. You can divide this process into four general tasks.

1. **Identify and characterize the problem domain**
   - Outline the process
   - Examine existing solutions
   - Identify key abstractions
   - Identify what parts of the process the framework will perform
   - Solicit input from clients and refine your approach
2. **Define the architecture and design**
   - Focus on how clients interact with the framework:
     - What classes does the client instantiate?
     - What classes does the client derive?
     - What functions does the client call?
   - Apply recurring design patterns
   - Solicit input from clients and refine your approach
3. **Implement the framework**
   - Implement the core classes
   - Test the framework
   - Ask your clients to test the framework
   - Iterate to refine the design and add features
4. **Deploy the framework**
   - Provide documentation in the form of diagrams, recipes, and especially sample programs
   - Establish a process for distribution
   - Provide technical support for clients
   - Maintain and update the framework

Developing Your Own Frameworks

Now that you have an understanding of the overall process and techniques, you can begin to develop your own frameworks. Here are some steps to help you get started:

1. Write a program with an existing framework.
2. Write a different type of program with the same framework.
3. Write a simple framework of your own and test it with small programs.
4. Write a program that combines your framework and an existing framework by using features of both.
5. Develop a small domain-specific framework.
6. Use your framework in more than one project.
7. Try to get other developers to use your framework and develop additional frameworks.

Deborah Adair, Taligent, Inc., 10201 North De Anza Boulevard, Cupertino, CA 95014-2233. 1-800-288-5545. Ms. Adair is a staff technical writer in the Taligent Technical Communications Department. She received a BS in Scientific and Technical Communications from the University of Washington.

Composition of Before/After Metaclasses in SOM

By Ira R. Forman, Scott Danforth, and Hari Madduri

In the IBM System Object Model (SOM), a class is a run-time object that defines the behavior of its instances by creating an instance method table. Because classes are objects, their behavior is defined by other classes called metaclasses. This article introduces and solves the problem of composing different before/after metaclasses in the context of SOM. An enabling element in the solution is SOM's concept of derived metaclasses; that is, at runtime a SOM system derives the appropriate metaclass of a class based on the classes of its parents and an optional metaclass constraint.

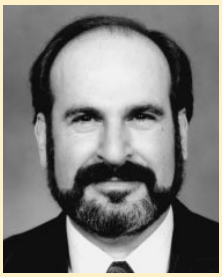
One way to view the history of programming is that progress is made by providing abstractions to ever larger entities and by ensuring the composability of those abstractions. In the beginning, assembly language instructions were gathered into control structures. Subsequently, control structures were gathered into procedures, and this was followed by gathering procedures into abstract data types. Now we have arrived at object-oriented programming, where abstract data types are gathered into an inheritance hierarchy.

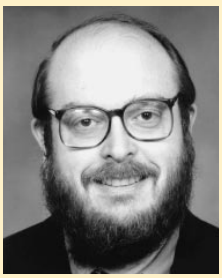
This article addresses a further abstraction—before/after metaclasses—and solves the problem of their composition. A before/after metaclass has a *before method* and an *after method* that are executed before and after the methods of the instances of its instances (an instance of a metaclass is a class, which in turn has instances). Applications for before/after metaclasses abound, including method tracing, invariant checking, path expression checking, and object locking. Given these opportunities, it is imperative that before/after metaclasses compose with one another.

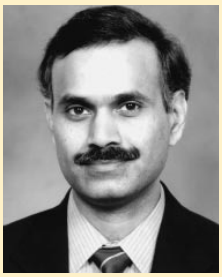
This article solves the before/after metaclass composition problem in the context of the IBM System Object Model (SOM). SOM is an object-oriented model [7,17,22,23]. The SOM runtime supports the model and allows programs written in arbitrary languages to use the model via the SOM Application Programming Interface (API). A binding is code that facilitates the use of a class or the implementation of a class. The SOMObject Toolkit[24] provides both usage and implementation bindings for C and C++. Language vendors for Smalltalk (Digitalk), COBOL (Micro Focus®), and C++ (IBM, Metaware, and Borland™) have also announced support for SOM in their products.

The SOM Model

In SOM, classes are objects whose classes are called *metaclasses*. A class is different from an ordinary object because a class has (in its instance data) an instance method table defining the methods to which instances of the class respond. During the initialization of a class object, a method is invoked on it that informs the class of its parents. This allows the class to build an initial instance method table. Once this is done, other methods are invoked on the class to override inherited methods or add new instance methods.


Ira R. Forman


Scott Danforth


Hari Madduri

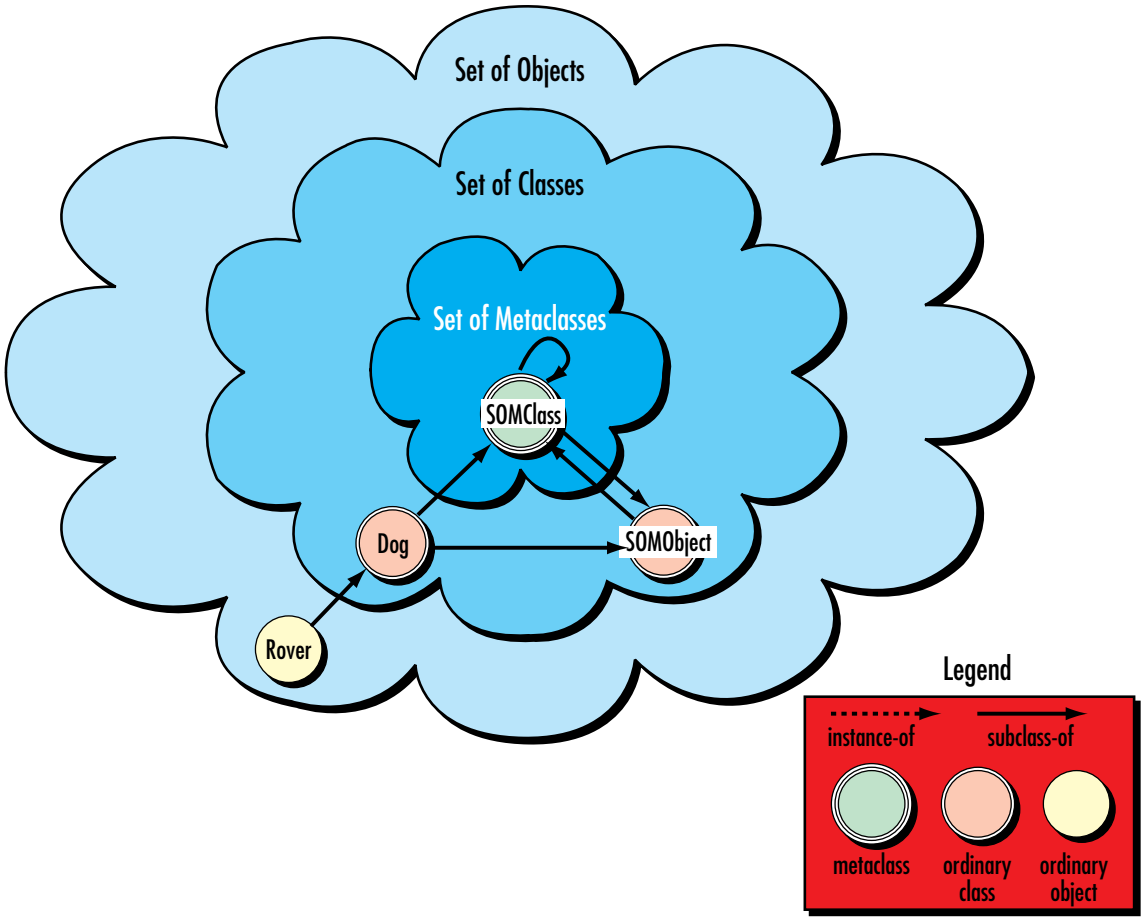


Figure 1. Structure of the SOM environment

When diagramming class hierarchies, this article uses the convention that metaclasses are drawn with three concentric circles; ordinary classes (classes that are not metaclasses) are drawn with two concentric circles; and ordinary objects (objects that are not classes) are drawn with a single circle. Figure 1 shows the initial state of an example SOM program. There are four objects: `SOMObject` (a class), `SOMClass` (a metaclass), `Dog` (an ordinary class), and `Rover` (an ordinary object). There are two relations among objects that must be understood:

1. There is the *instance of* relation between objects and classes depicted by the dashed arrow from an object to its class. When convenient, the inverse relation *class of* is also used. `SOMObject` is an instance of `SOMClass` and `SOMClass` is the class of itself. An object's class is important because an object responds only to the methods that are supported by its class (that is, the methods that the class introduces or inherits).
2. There is a relation between classes called the *subclass of* relation, which is depicted by the solid arrow from a class to each of its *parents*. `SOMClass` is a subclass of `SOMObject`. `SOMObject` has no parents.

`SOMObject` introduces the methods to which all SOM objects respond. In particular, `SOMObject` introduces the `somDispatch` method. This method provides a single, general dynamic dispatch mechanism for executing method calls on objects. A class can arrange its instance method table so that all method calls are routed through `somDispatch`. As a result, it is simple for SOM metaclass programmers to arrange for completely arbitrary processing in connection with method invocations on SOM objects.

As a subclass of `SOMObject`, `SOMClass` is an object but it also introduces the methods to

which all classes respond. For example, SOMClass introduces the somNew method, which creates instances of a class. The methods responsible for creating and modifying instance method tables are also introduced. All metaclasses in SOM are ultimately derived from SOMClass. (Similar arrangements of classes are also used in CLOS[12], ObjVlisp[5], Dylan[2], and Proteus[21].) With the SOM API, you can create new abstractions by programming metaclasses. In more general terms, this has been referred to as a metaobject protocol[12,13] or computational reflection[15]. The strength of this general approach is that new abstractions can be created after the object model is implemented. That is, the before/after metaclasses are not part of the SOM kernel, rather they are part of a framework for programming metaclasses[9] that is built with the SOM API. Because SOM provides a metaobject protocol, we were able to add a new abstraction to SOM.

Interfaces to SOM objects are described using IDL, an object interface definition language defined by the Common Object Request Broker Architecture (CORBA[16]) standard of the Object Management Group (OMG). SOM IDL is a CORBA-compliant version of IDL used to allow SOM class descriptions to be supplied in addition to object interface definitions. (That is, the interface to a class is described by the IDL alone; SOM IDL allows additional information about the implementation to be added.) The SOMObjects Toolkit has tools called *emitters* that translate SOM IDL into language-specific bindings for the corresponding classes of SOM objects (For C programmers, this means that emitters produce header files for both the users of the class and the implementor of the class).

Figure 2 shows the basic structure of an IDL definition for an object interface named Dog. At the same time, it is a SOM IDL description of a class Dog that supports this interface. The #ifdef and #endif (omitted from subsequent examples) are part of the IDL language and are used to hide the SOM class implementation section from non-SOM IDL compilers.

In Figure 2, the interface Dog inherits from the SOMObject interface; at the same time, the class Dog is declared to be a subclass of SOMObject. CORBA and SOM support multiple inheritance; additional parents of Dog can be listed alongside SOMObject in a comma-separated list. The actual methods and instance variables of Dog are not relevant to the current discussion.

```
interface Dog : SOMObject
   {
    method and attribute declarations here
    #ifdef  __SOMIDL__
     implementation
   {
      metaclass = SOMClass;
         instance variable declarations here
       };
    #endif
};
```

Figure 2. IDL definition for object interface Dog

The implementation section can explicitly indicate a metaclass to be associated with the class of objects that support the interface being defined. This association is not necessarily direct. For reasons that will become clear, the actual class of the class described by any given SOM IDL is generally a subclass of the indicated metaclass.

Before/After Metaclasses

A *before method* is a behavior that precedes the action of some program construct. An *after method* is a behavior that succeeds the action of some program construct. Before and after methods are familiar to users of CLOS[12,19] where the granularity of application is the individual method. In class-based object models such as SOM, the more natural granularity for before/after methods is the class, because there are many applications that fit this granularity (see next section).

The SOMMBeforeAfter metaclass therefore introduces two methods—BeforeMethod and AfterMethod —arranged by its instances (classes) to run respectively before and after each instance method. By default, these two methods do nothing. To define a specialized before/after behavior, it is necessary to create a subclass of SOMMBeforeAfter and override the BeforeMethod and the AfterMethod with the desired behavior.

In Figure 3, the Barking metaclass overrides BeforeMethod and AfterMethod with a method that makes a "woof" sound when executed. As a result, all methods supported by the class BarkingDog (an instance of Barking) have this before/after behavior. That is, the object Lassie goes "woof" before and after each method invoked on it runs, because it is an instance of BarkingDog. The IDL for the BarkingDog class is given at the right of the figure. The IDL is the

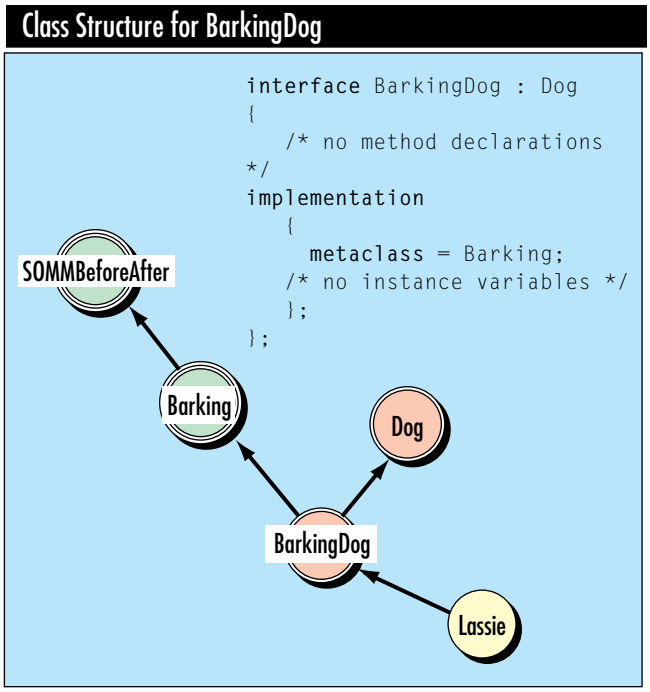


Figure 3. BarkingDog class structure

```
somDispatch ( self, primaryMethod, ... )
BeforeMethod( class(self),
              self,
              primaryMethod, ... );
retval := primaryMethod( self, ... );
AfterMethod( class(self),
             self,
             primaryMethod,
             retval, ... );
return retval;
```

Figure 4. New dispatcher for SOMObject

only source code that needs to be written; the compiler of the SOMObjects Toolkit can generate all the necessary code from IDL to implement `BarkingDog` (in either C or C++).

The essence of the workings of `SOMMBeforeAfter` is a method that overrides the method `somDispatch` introduced by `SOMObject`*. The new dispatcher looks like Figure 4.

In Figure 4, `primaryMethod` is the method being invoked on a target object, such as `Lassie`, for which before/after behavior is desired. In the pseudo-code used in this article, the first parameter to a method invocation is always the target object. The ellipses represent all the other actual parameters to the method. As noted earlier, the metaclass of the object `Lassie` supports `BeforeMethod`; that is, the class `BarkingDog` responds to `BeforeMethod`. This is why, in the above pseudo-code, `class(self)` is the target object for the `BeforeMethod` and `AfterMethod` method invocations.

Usefulness of Before/After Metaclasses

Before attacking the composition problem of before/after metaclasses, consider their usefulness. As mentioned earlier, CLOS has the notion of before/after methods, but our experience indicates that the more useful granularity for a class-based object model is the class. Foote and Johnson[10] advocate class-level granularity. Pascoe[20] does also, but his encapsulators apply to all method invocations on a class instance rather than a class. This is also the granularity used by the Demeter[14] system, which does the implementation by source code transformation.

Software engineering of classes has many examples of uses of before/after methods. Method tracing is a primary example of a useful software engineering tool that fits naturally into the before/after paradigm at the class granularity. Another example is invariant checking. Imagine a metaclass that checks the invariant supplied by the class programmer as a method on the class. In addition to its reusability, this type of metaclass would have the advantage of ensuring that the invariant is checked when new methods are added to the class. Other types of verification and monitoring, such as path expressions[4] or behavioral expressions[1], are also feasible.

Concurrency yields other opportunities to use before/after metaclasses. For example, atomicity can be factored into a metaclass; the before method acquires a semaphore and the after method releases the same semaphore. In addition, we have found that before/after metaclasses provide a convenient way to offer framework capabilities to customers. The SOMObjects Toolkit contains a framework for creating replicated objects[24]; this framework has a set of rules

* Those readers not familiar with SOM may skip this note. For the sake of efficiency, methods in SOM are usually invoked directly and the `somDispatch` method is not generally called. In this scheme the `SOMMBeforeAfter` metaclass arranges for `somDispatch` to be invoked by placing stubs in the method table to call `somDispatch` (this capability is part of the SOM API). The `SOMMBeforeAfter` metaclass also arranges that the contents of the original method table be saved so that `somDispatch` can invoke the primary method. In addition, the `SOMMBeforeAfter` metaclass ensures that `somDispatch` does not dispatch itself (which would cause a dispatch loop). The details of how this is done are very specific to the SOM API and beyond the scope of this article. See reference 24 for more information SOM API.

for conveying the replicated property to a class. The rules require locking a set of replicas before an update, followed by the propagation and unlock after the update (the objective is to ensure one-copy serializability). The majority of the work required by this set of rules can be done by a before/after metaclass. We have used `SOMMBeforeAfter` to construct metaclasses for both tracing and replication (see Figure 5).

Other frameworks in our toolkit could similarly be aided. Elsewhere, Paepcke[18] provides a detailed example of how to use a metaclass in CLOS to make a class persistent. Suppose you wish to provide a class library that has `n` classes. In addition, suppose there are `p` properties that must be included in all combinations for all classes. Potentially, the library must have $n2^p$ classes.

If we hypothesize that (fortunately) all these properties can be captured by before/after metaclasses, the size of the library is `n+p`. The users of the library need only produce those combinations necessary for their applications. This problem is one faced by users of some object-oriented databases and has also arisen in the design of the OMG Persistence Standard[6]. In this situation, one obvious conclusion is unavoidable: before/after metaclasses are not useful unless they compose, because if not, the use of one before/after metaclass would preclude the use of others. (What good is a trace metaclass if it cannot be used to debug instances of the replicable metaclass?)

The Composition Problem

Suppose there are before/after metaclasses `Barking` (as before) and `Fierce` (shown in Figure 6), which has a `BeforeMethod` and `AfterMethod` that both "growl"—both make a "grrrr" sound when executed. We can now create a `FierceDog` or a `BarkingDog`, but we still have not addressed the question of how to compose the properties of fierce and barking.

Composability means having the ability to easily create a `FierceBarkingDog` that goes "grrr woof woof grrr" when it responds to a method call, or a `BarkingFierceDog` that goes "woof grrr grrr woof" when it responds to a method call.

Composing the properties of fierce and barking is complicated because there are several ways in which these compositions can be expressed. Figure 7 shows three techniques in which a programmer might naturally indicate such a composition. These are labeled Technique 1, Technique 2, and Technique 3, which create the `FierceBarkingDog` classes named `FB-1`, `FB-2`, and `FB-3`, respectively. The SOM IDL for each class is given above a diagram that depicts the context in which the class description is given.

SOM 2.1 Metaclass Framework

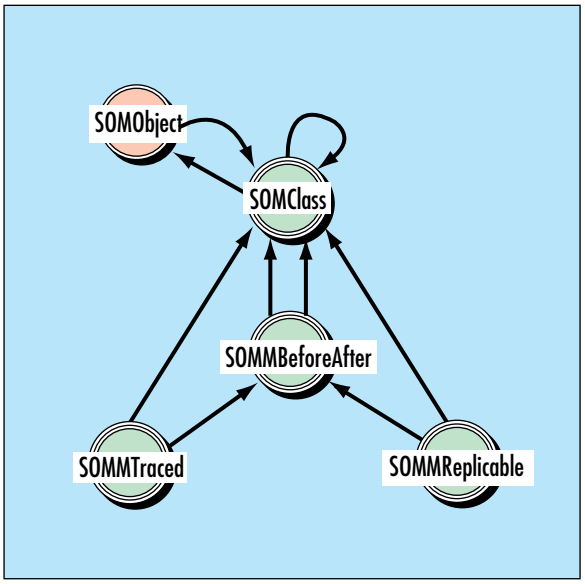


Figure 5. SOM 2.1 metaclass framework

Fierce and Barking Metaclasses

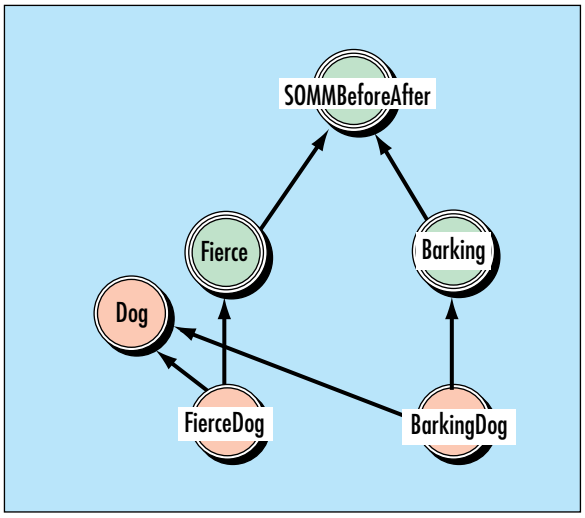


Figure 6. Fierce and Barking metaclasses

- **Technique 1**: A new metaclass (`FierceBarking`) is created with both `Fierce` and `Barking` as parents; an instance of this new metaclass (`FB-1`) should be a `FierceBarkingDog` (if `Dog` is a parent).

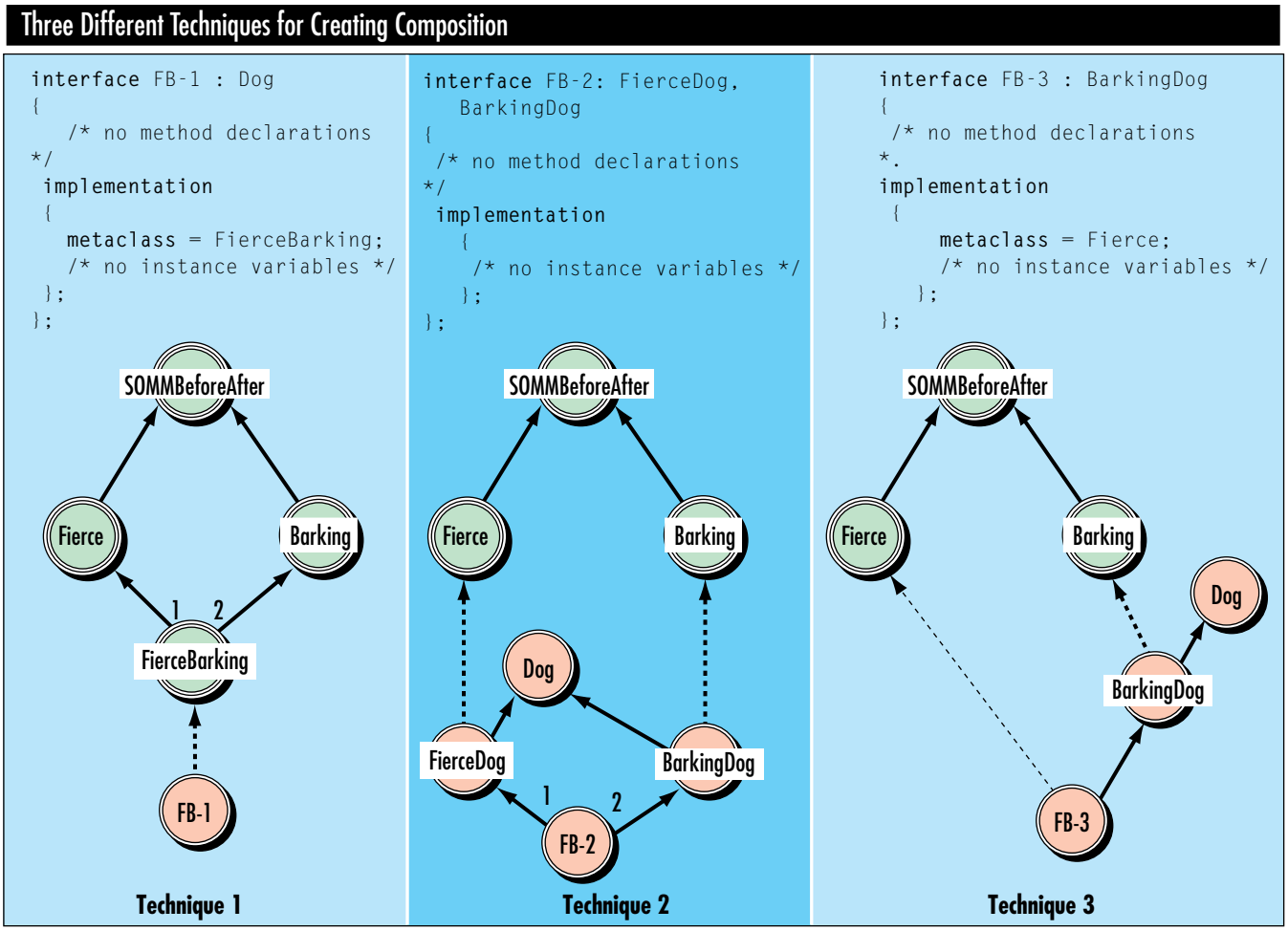


Figure 7. Three different techniques for creating composition

- **Technique 2**: A new class is created that has parents that are instances of `Fierce` and `Barking` respectively; that is, `FB-2` should be a `FierceBarkingDog` too (assuming `FierceDog` and `BarkingDog` do not further specialize `Dog`).
- **Technique 3**: `FB-3`, which should also be a `FierceBarkingDog`, is created by declaring that its parent is a `BarkingDog` and that its explicit (syntactically declared) metaclass (drawn with the light dashed arrow) is `Fierce`.

These three techniques should produce the same result; that is, `FB-1`, `FB-2`, and `FB-3` should be equivalent classes—behave the same and have instances that behave the same. That is because composition of metaclasses must be easily understood by programmers. Non-equivalence of these three techniques would certainly lead to a system in which programming is complex and error-prone. This conclusion leads us to ask what common property these techniques have upon which an equivalence can be based. There is such a property and SOM provides special support for it.

The Derived Metaclass in SOM

SOM allows and encourages the definition and explicit use of metaclasses. At the same time, however, SOM relieves programmers of the responsibility *for getting the metaclass right* when defining a new class. At first glance, this might seem to be merely a useful (though very important) convenience; but, in fact, it is absolutely essential in SOM. This is because SOM is predicated on binary compatibility with respect to changes in class implementations. Even though programmers might, at one time, know the metaclasses of all classes above a new subclass, and as a result, be able to explicitly derive an appropriate metaclass for the new class, SOM must guarantee that this new class still executes

correctly when any of its ancestor class' implementations are changed (and this could include a choice of different metaclasses). SOM programmers never need to consider a newly defined class' ancestors' metaclasses. Instead, explicit metaclasses should be used only to *add in* desired behavior for a new class. Anything else that is needed is done automatically [12].

Figure 8 shows a simple single-inheritance example. `A` is an instance of `AMeta`. We assume that `AMeta` supports a method bar and that `A` supports a method `foo` that uses the expression `bar ( class( self ) )`. That is, the method `foo` invokes a method on the class of the object on which `foo` is operating.

Now consider what happens when `A` is subclassed by `B`, a class that has an explicit metaclass declared in its SOM IDL. If the class hierarchy were to be formed as in Figure 8, an invocation of `foo` on an instance of `B` would fail because `BMeta` does not support bar. This situation is called *metaclass incompatibility*. Since SOM does not allow hierarchies with metaclass incompatibilities, it builds derived metaclasses that prevent this problem from occurring.

The actual SOM class hierarchy that results for `B` is depicted in Figure 9, where SOM has automatically built the metaclass `DerivedMetaclass`. This ensures that the invocation of `foo` on instances of `B` do not fail. This example shows that the metaclass statement in the SOM IDL is treated as a constraint on the actual metaclass. The derived metaclass can be viewed as the minimal metaclass supporting the constraints of metaclass compatibility.

Although other articles[3,11] have called this situation the *metaclass compatibility problem*, none go beyond a characterization of the compatibility condition required on the metaclass statement. In SOM there is no such problem; in a situation where the explicitly declared metaclass is not compatible with the parents of the class, an appropriate metaclass is constructed—this is the derived metaclass. Because class construction is a dynamic activity in SOM, this derivation is actually accomplished at runtime with no need for prior description in IDL*.

The Composition Solution

In Figure 7, the derived metaclass constructed by SOM for `FB-3` will be `FierceBarking`, not `Fierce`

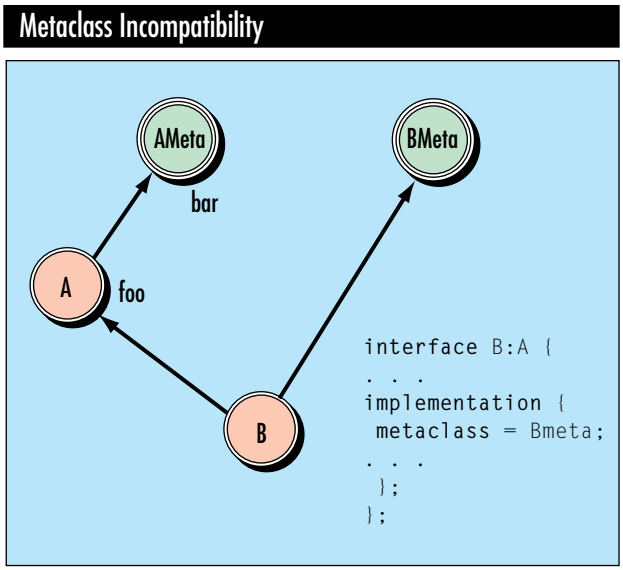


Figure 8. Example of a metaclass incompatibility

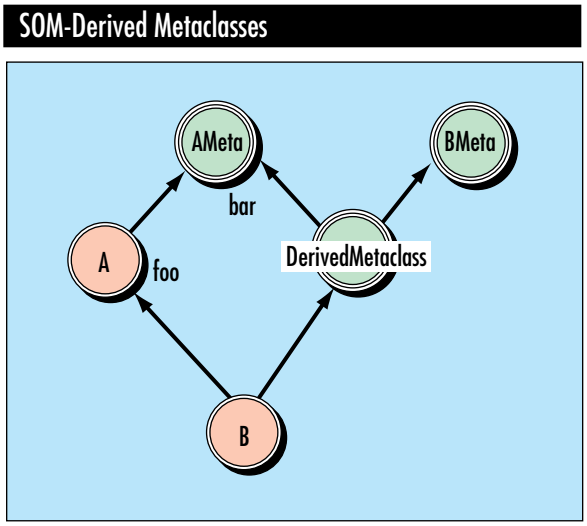


Figure 9. SOM-derived metaclasses prevent incompatibilities

as indicated in the SOM IDL. Although the metaclass of `FB-2` in Technique 2 is not explicitly declared, the derived metaclass provided by SOM for `FB-2` will be `FierceBarking`.

Figure 10 combines the diagrams in Figure 7 and shows the actual class relationships, which are established when the class objects are instantiated. The explicit metaclass in the SOM IDL of `FB-1` is its derived class `FierceBarking`. The derived metaclass of `FB-2` is also `FierceBarking`; however, the derived metaclass of `FB-3` is not

* CLOS does not allow the construction of metaclass incompatibilities. When a class is constructed validate-superclasses is called to ensure that all superclasses have the same metaclass as the class being constructed (see references 19 and 12, pages 84 and 240-241 respectively); if this condition is not true, an error is signaled.

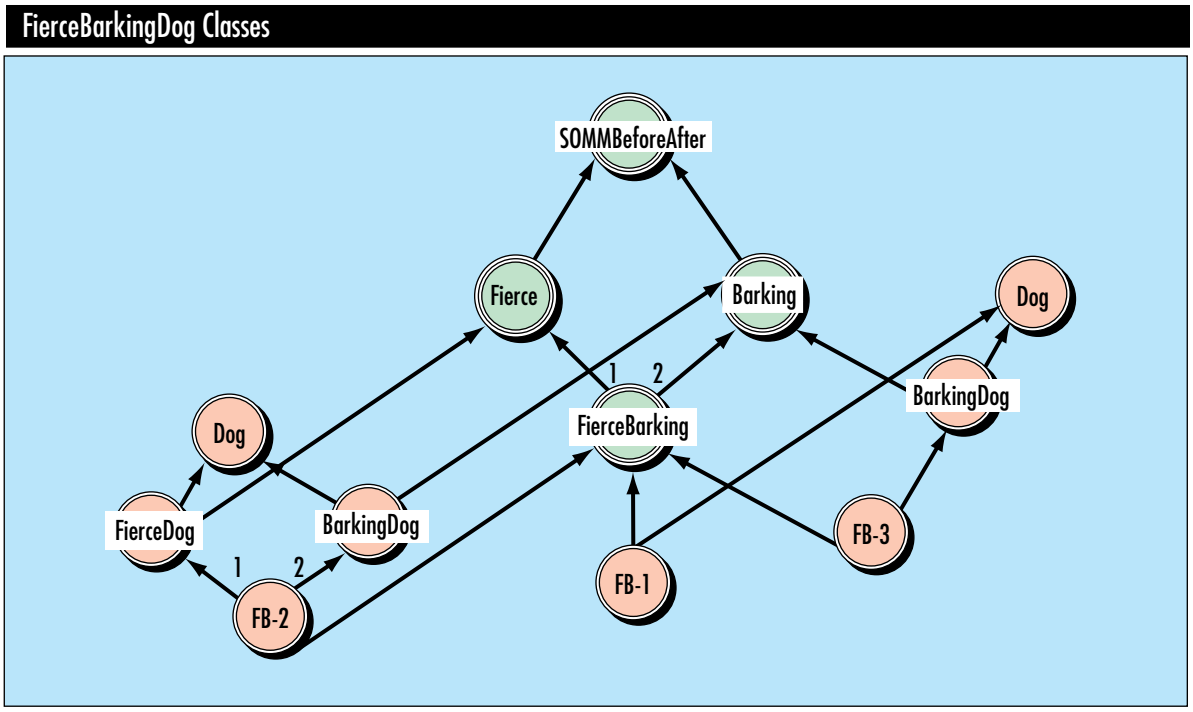


Figure 10. FierceBarkingDog classes have equivalent metaclasses

```
somDispatch ( self, primaryMethod, ... )
BeforeMethodDispatch( class( class ( self )),
                      self, ... );
retval := primaryMethod( self, ... );
AfterMethodDispatch( class( class ( self )),
                     self, ... );
return retval;
```

Figure 11. Main dispatch function

```
BeforeMethodDispatch( aMetaclass,
                      clientObject, ... )
if  aMetaclass defines BeforeMethod
       BeforeMethod( clientObject, ... )
else
       for all parents of aMetaclass
                        that support BeforeMethod
              BeforeMethodDispatch( aParent,
                                    clientObject,
                                    ... )
```

Figure 12. BeforeMethodDispatch implementation

the explicit metaclass in the SOM IDL. It too is `FierceBarking`.

The solution to the composition problem is apparent. The common element among the three techniques for expressing before/after composition is the metaclass `FierceBarking`, which is the class of `FB-1`, `FB-2`, and `FB-3`. In `FB-1`, the relationship is explicit and in `FB-2` and `FB-3` the relationship is derived. Therefore, we conclude that composition can be based on the "completed" metaclass hierarchy that results from using derived metaclasses in SOM.

By the nature of before/after, the main dispatch function now looks like Figure 11.

The `primaryMethod` is the identifier of the method being invoked by the application and acquiring before/after behavior. To produce an appropriate composed order of before method invocations, `BeforeMethodDispatch` does a preorder traversal of the metaclass hierarchy towards `SOMClass`, looking for the first metaclass on each path that defines `BeforeMethod`. Each such `BeforeMethod` is then invoked on the client object. The `BeforeMethodDispatch` implementation looks like Figure 12.

The search restriction to metaclasses that support `BeforeMethod` is based on the fact that a before/after metaclass can have parents that are not before/after metaclasses (parents that are not

descendents of SOMMBeforeAfter). If the metaclass of the client object defines a BeforeMethod, the search succeeds and only one BeforeMethod is called. The rationale for this is simple: the creator of a before/after metaclass knows the parents of this metaclass, and when overriding BeforeMethod, can be expected to explicitly invoke any inherited functionality that is necessary (using parent method calls).

The AfterDispatchMethod is very similar, except that the parents are traversed in the reversed order. Figure 13 shows the AfterDispatchMethod implementation.

```
AfterMethodDispatch( aMetaclass,
                     clientObject, ... )
if  aMetaclass defines AfterMethod
       AfterMethod(  clientObject, ... )
else
       for all parents of aMetaclass
                      that support AfterMethod
                     (in reverse order)
              AfterMethodDispatch( aParent,
                                   clientObject,
                                   ... )
```

Figure 13. AfterDispatchMethod implementation

Loose Ends

The before/after metaclass facility described in this article has been available inside IBM since August 1993; for example, the traced metaclass is used in parts of the SOMObjects Toolkit (Version 2.0). The facility became generally available as part of SOMObjects Toolkit (Version 2.1) in November 1994. Before concluding, there are several issues to address, which for ease of presentation have been ignored until this point.

First, our solution to the composition of before/after metaclasses has a pleasant property: composition is associative. Figure 14 shows three before/after metaclasses—A, B, and C—that introduce three before methods respectively (Before1, Before2, and Before3). Metaclass F represents the composition (AB)C and G represents the composition A(BC). Both compositions lead to the same sequence of before method invocations (Before1; Before2; Before3). Of course, composition is not commutative; for example, a FierceBarkingDog is not the same as BarkingFierceDog (which goes "woof grrr grrr woof"). The order of the metaclasses depends on the search order, which is determined by the order of the parents.

Second is the issue of exempting a method from the before/after behavior. We have seen situations for different before/after methods that range from exempting a particular primary method to exempting a particular kind of method, such as read-only, to exempting all methods inherited from a particular parent. We have found that the simplest solution is to have the designer of the particular before/after metaclass determine the scheme for method exemptions. Usually the metaclass designer introduces a predicate method to be called by the BeforeMethod and the AfterMethod. If the

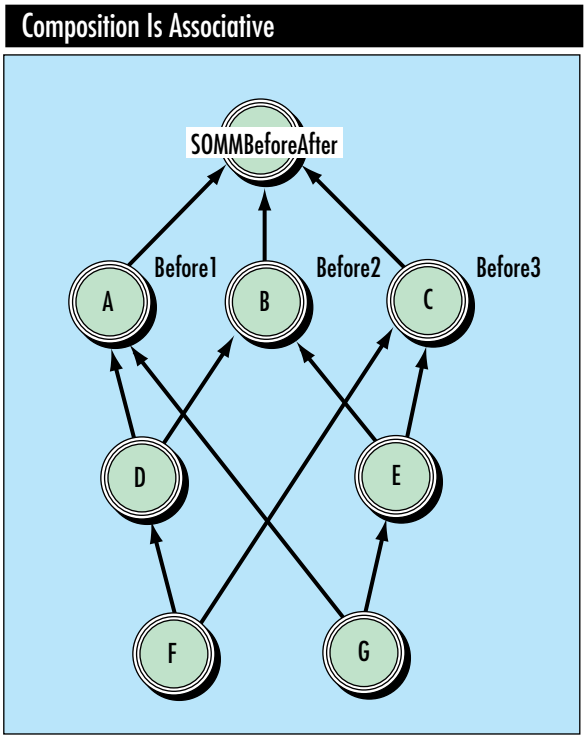


Figure 14. Composition is associative

predicate returns true, both wrapper methods return without doing anything.

Third, for the sake of simplicity, the before (after) dispatch method (presented above) does not check whether the before (after) method has already been executed during the search of the metaclass hierarchy. For example, if in Figure 14, an H is added as a subclass of both F and G, all the before methods are executed twice. Several techniques can solve this problem. So far, no case has arisen where it is desirable to have multiple executions of before/after methods.

Another issue is having the before method determine that the primary method should not be

run. To handle this case, the before method is extended to return a boolean to the dispatcher, which contains "true" if the primary method is to be run. If the primary method is not run, neither is the corresponding after method. Because the before and after methods belong to the same metaclass, the before method can do any tidying up the after method might have been required to perform (this eliminates the need to communicate to the after method that the primary method was not invoked).

Conclusions

The central issue addressed by this article is raising the level of programming by composing metaclasses. The standard notion of inheritance-based subclassing represents a union-like operation for composition of class implementations. There is no reason to believe that one such operation is sufficient for all the possible compositions that need to be performed. With the approach described here, we believe that significant object properties can be implemented using before/after metaclasses and that these properties can be subsequently composed.

Linguistically, a property is often represented by an adjective while a class is represented by a noun. Composition of metaclasses should be as easy as putting a sequence of adjectives in front of a noun when we speak.

Acknowledgements

Mike Connor and Larry Raper are the designers of the SOM model and API; their insight in providing SOM with metaclasses provides the basis upon which we worked. Andy Martin did an outstanding job implementing the first SOM compiler. We wish to thank Liane Acker, Gregor Kiczales, and Harold Ossher for their comments on earlier drafts of this article. In addition, the comments of the OOPSLA referees were very valuable and greatly appreciated.

References

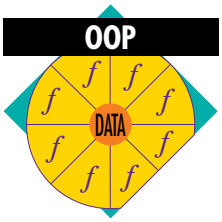
1. Atkinson, C. *Object-Oriented Reuse, Concurrency, and Distribution: An Ada-based Approach*. Addison-Wesley, 1991.
2. Apple Computer, Inc. *Dylan: An Object-Oriented Dynamic Language*. 1992.
3. Briot, J. P. and Cointe, P. "Programming with Explicit Metaclasses in Smalltalk-80." *OOPSLA '89 Conference Proceedings* (October 1-6, 1989). p. 419-431.
4. Campbell, R.H. and Habermann, A.N. "The Specification of Process Synchronisation by Path Expressions." *Lecture Notes in Computer Science* (16). Springer-Verlag, 1974. p. 89-102.
5. Cointe, P. "Metaclasses are First Class: the ObjVlisp Model." *OOPSLA '87 Conference Proceedings* (October 4-8, 1987). p. 156-165.
6. Copeland, G. Private communication. 1994.
7. Danforth, S. "A Bird's Eye View of SOM." *IBM OS/2 Developer* (Winter 1992).
8. Danforth, S. and Forman, I.R. "Derived Metaclasses in SOM." *Proceedings of the 1994 Conference on Technology of Object-Oriented Languages and Systems*. Versailles, France. (April 1994).
9. Danforth, S. and Forman, I.R. " Reflections on Metaclass Programming in SOM." *OOPSLA '94 Conference Proceedings* (October 23-27, 1994).
10. Foote, B. and Johnson, R.E. "Reflective Facilities in Smalltalk-80." *OOPSLA '89 Conference Proceedings* (October 1-6, 1989). p. 327-335.
11. Graube, N. "Metaclass Compatibility." *OOPSLA '89 Conference Proceedings* (October 1-6, 1989). p. 305-316.
12. Kiczales, G., des Rivieres, J., and Bobrow, D. G. *The Art of the Metaobject Protocol*. Cambridge, Massachusetts: The MIT Press, 1991.
13. Kiczales, G. and Paepcke, A. *Open Implementations and Metaobject Protocols*. Cambridge, Massachusetts: The MIT Press, 1994.
14. Lieberherr, K.J. and Xiao, C. "Object-Oriented Software Evolution." *IEEE Transactions on Software Engineering* 19 (April 1993). p. 313-343.
15. Maes, P. "Concepts and Experiments in Computational Reflection." *OOPSLA '87 Conference Proceedings* (October 4-8, 1987). p. 147-155.
16. Object Management Group. *The Common Object Request Broker: Architecture and Specification*. OMG Document Number 91.12.1 Revision 1.1.
17. *OS/2 Technical Library System Object Model Guide and Reference* (S10G6309). Armonk, New York: IBM Corporation, 1991.
18. Paepcke, A. "PCLOS: A Critical Review." *OOPSLA '89 Conference Proceedings* (October 1-6, 1989). p. 221-237.

19. Paepcke, A. (ed.). *Object-Oriented Programming: The CLOS Perspective.* Cambridge, Massachusetts: The MIT Press, 1993.
20. Pascoe, G.A. "Encapsulators: A New Software Paradigm in Smalltalk-80." *OOPSLA '86 Conference Proceedings* (September 29 - October 2, 1986). p. 341-346.
21. Russinoff, D.M. "Proteus: A Frame-Based Nonmontonic Inference System." *Object-Oriented Concepts, Databases, and Applications.* Kim, W. and Lochovsky, F.H. (ed.) New York: ACM Press, 1989. p. 127-150.
22. Sessions, R. and Coskun, N. "Object-Oriented Programming in OS/2 2.0." *IBM Personal Systems Developer* (Winter 1992).
23. Sessions, R. and Coskun, N. "Class Objects in SOM." *IBM OS/2 Developer* (Summer 1992).
24. *SOMObjects Developers Toolkit—User's Guide and Reference Manual.* Armonk, New York: IBM Corporation, 1993.

Ira R. Forman, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Forman is a member of the Object Technology Products Group, where he specializes in object-oriented distributed systems and object composition. He has a PhD in Computer Science from the University of Maryland.

Scott Danforth, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Danforth is currently developing language-neutral object technology for binary class libraries and system-level frameworks for OOP. He has a PhD in Computer Science from the University of North Carolina at Chapel Hill.

Hari Madduri, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Madduri, the lead designer of Object Replication Framework is currently the architect/designer of distributed SOM. Mr. Madduri has a PhD in Computer Science from the University of Wisconsin in Madison.



CommonPoint Frameworks Take the Spotlight

By Joyce Uggla

Since its founding in 1992, Taligent, Inc., has been an avid evangelist of object-oriented frameworks. That is because this independent software company—owned by Apple Computer, Inc., Hewlett-Packard Company, and IBM Corporation—is betting its future on this technology by building an object-oriented-based product that promises to deliver dramatic developer productivity and innovation, distributed enterprise application support, and a superior user experience.

Taligent's CommonPoint™ application system is operating system independent and supports the creation and deployment of distributed applications. Application systems combine the common functions (such as text editing, graphics, multimedia, and compound document features) built into today's monolithic applications with common operating system functions (like networking, communications, and data access) to provide developers with a portable foundation to build distributed applications and software components.

"Application systems are an important emerging software technology that will help to re-ignite application innovation," said David Cearley, vice president and director of Workgroup Computing Strategies at META Group™, Inc., a market research firm in Stamford, Connecticut. "Application systems eliminate the redundancies in today's monolithic applications and operating system platforms and provide application portability across operating systems. Taligent's CommonPoint architecture is well positioned to compete in this market."

Earlier in 1995, Taligent shipped a final beta reference release of the CommonPoint system to its investors—Apple, HP, and IBM—and to its early developers. Since then, the company has extended its beta program to a new set of developers and is moving closer to its target ship date of mid-1995.

"Taligent's goal is to establish the CommonPoint application system as the object-oriented standard for the next generation of enterprise applications," said Joe Guglielmi, chairman and CEO of Taligent. "Delivery of the beta reference release is an important milestone in accomplishing this goal because it allows our investors and early partners to continue their development efforts with a feature-complete version of CommonPoint. We plan to move forward aggressively from here to deliver the final reference release this summer."

Why an Application System?

"Application system" is a new term for a class of software that focuses on application development independent of underlying hardware or software. It represents the intersection of several converging trends:

- **Increasing abstraction**. The trend is moving programmer attention upward from hardware details—from disk sectors to files to databases; from memory addresses to variables, arrays, and structures to objects. By working with entities that map one-for-one with elements of the problem, developers become more effective.

Joyce Uggla

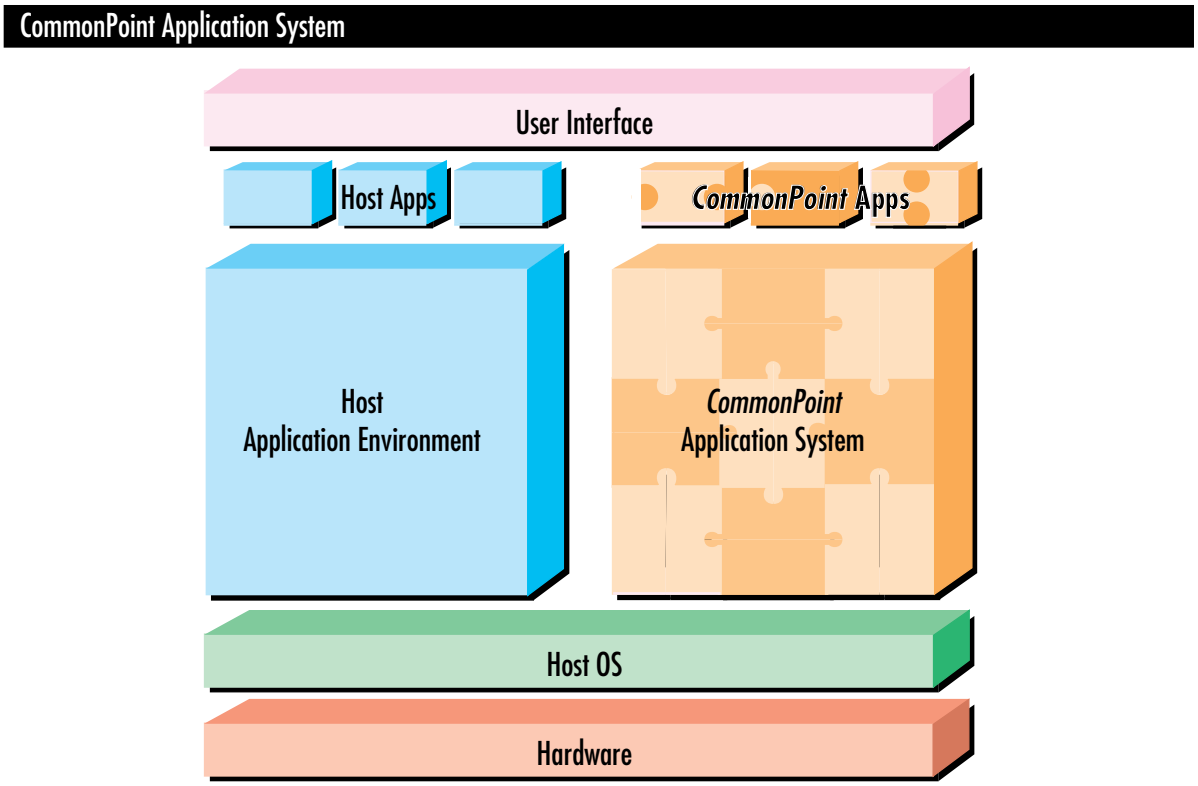


Figure 1. CommonPoint application system

- **Platform independence**. Today UNIX is a cross-platform operating system. SQL is a common query language for relational databases from disparate vendors. Open Database Connectivity (ODBC) is a vendor-neutral database programming interface.

  Platform independence provides rapid porting of applications and support for applications distributed across heterogeneous platforms and networks.

- **Opening the system**. Plug-ins are available for PhotoShop, Excel, and other applications. Open systems allow competition in replaceable parts and innovation in both general-purpose and very focused areas, with special-purpose extensions able to satisfy very specific needs.

- **Accommodating the user**. Graphical window system interfaces initiated a new era in user accommodation. The computer pictured items from the users' world, such as trash cans and folders, and allowed users to manipulate them in familiar ways.

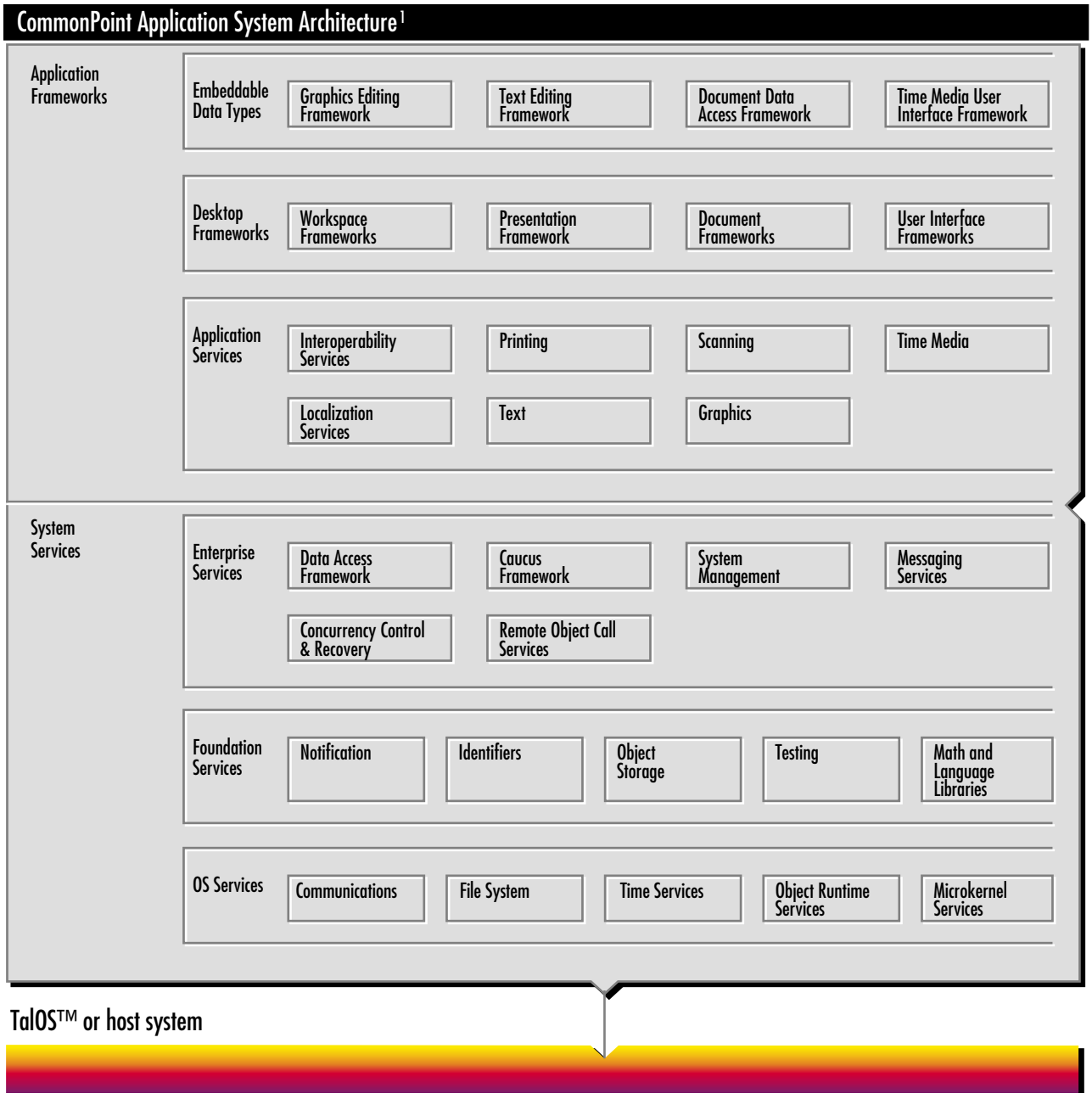
  Computers are getting easier to use because developers are understanding that successful products accommodate the user rather than the reverse.

Application systems bring these trends together and provide developers with high-level, platform-independent abstractions that cover all common application functions. An application system defines a user model and provides a user interface that embodies the model. The best user models let the user work in the user's way, rather than requiring the learning of special computer techniques.

A comprehensive application system is rich in functionality, but maintains consistency both internally and with third-party and application code.

Although other products have elements of an application system, the CommonPoint product is the first designed specifically as an application system (shown in Figure 1). It consists of a rich set of extensible frameworks, utilities, and an innovative enterprise graphical user interface.

Specifically, more than 100 object-oriented frameworks (700,000 lines of C++ code, and just under 2,000 classes) make up the CommonPoint system, which provides rich functionality for compound documents, collaboration, multimedia,



[1]Features and functions shown are subject to change by Taligent without notice. Some functions are not in the first release.

Figure 2. Architecture for CommonPoint Application System

2-D and 3-D graphics, data access, communications, and distributed objects. That means developers can leverage this functionality, so they do not have to build it into each application.

Initially, the CommonPoint application will run on top of existing 32-bit operating systems such as AIX, OS/2, HP-UX®, and a future version of Mac™ OS. Taligent also has plans to offer a version of the CommonPoint system for Windows NT and Windows '95.

CommonPoint Architecture

The CommonPoint application system provides the developer with both breadth and depth of programming functionality. Figure 2 offers an overview of the key facilities available in the system.

At its highest level, the CommonPoint application system can be thought of as providing two distinct sets of services: a set of application frameworks that can be used to create powerful,

interactive applications; and a set of system services that can be used to manipulate data, communicate with other computers, and interface to the underlying operating system.

The application frameworks include the following:

- **Embeddable data types** support viewing and editing of complex data types.
- **Desktop frameworks** provide support for the CommonPoint application model, including its user interface policy, its "look and feel," and its compound document architecture.
- **Application services** support media and data handling services needed to create industrial-strength interactive applications.

The depth of the system can be illustrated by the text frameworks, which include frameworks for line layout, paragraph styles, text styles, text and style storage management, and character sets. The graphics frameworks include 2-D and 3-D graphics, colors, font support, sprites, pixel buffers, graphic devices, and displays.

The system services includes the following components:

- **Enterprise services** is a set of services that allow the CommonPoint application system to interoperate with other computers distributed within an enterprise.
- **Foundation services** is a fundamental set of object services that make it easier to write object-oriented programs.
- **OS services** provide basic support for creating programs that work across a wide variety of host operating systems and hardware platforms.

Taligent's early developers are already recognizing the benefits of the system. "CommonPoint is our product strategy choice because it enables us to build a new class of collaborative applications," said Stephan Adams, president of Adamation, Inc., a developer of commercial and custom enterprise applications based upon object-oriented platforms. "This functionality makes CommonPoint distinctive in the marketplace, thus providing unique value to customers and new opportunities for developers. The collective frameworks allow us to concentrate on the components of our applications rather than the low-level functions."

Developer Productivity with Frameworks

Frameworks and systems based on frameworks, such as the Taligent environment, empower developers to fully realize the potential of improved design and code reuse, including reduced development requirements, reduced maintenance, and higher reliability.

What are Frameworks?

A *framework* is a set of cooperating classes that constitute a generic design solution that can be adapted to a variety of specific problems within a given domain.

In the same way that classes are aggregates of functions and data, frameworks are aggregates of classes—but frameworks are not mere collections of classes. Frameworks are architectural; they provide infrastructure, which reduces the amount of code that you must write and thereby alleviates many productivity problems. Frameworks represent the next level of abstraction beyond class libraries, just as classes represent the next level of abstraction beyond functions and data.

Frameworks and systems based on frameworks empower developers to fully realize the potential of improved design and code reuse.

Capturing Expertise

A framework represents a generic design solution. It is a meta-solution that represents the set of all possible solutions within a particular problem domain, not any one solution.

A framework abstracts the essential entities, state, and behavior in its problem domain. It provides key mechanisms, defines the interaction protocols for key scenarios, and encapsulates and enforces fundamental invariants. It has strong "wired-in" connections among its objects that capture design decisions common to its problem domain.

In this manner, a framework embodies the domain expertise of the designer of the framework—it captures the programming expertise necessary to solve a particular kind of problem. See Figure 3.

Using Frameworks

A framework dictates the architecture of applications based on it. It defines an application's overall structure, its partitioning into classes and objects, the key responsibilities, how the classes and objects collaborate, and the flow of control. A framework also defines and enforces the responsibilities of a developer who wants to make use of the framework, as well as the degrees of freedom available to a developer who wants to customize the framework.

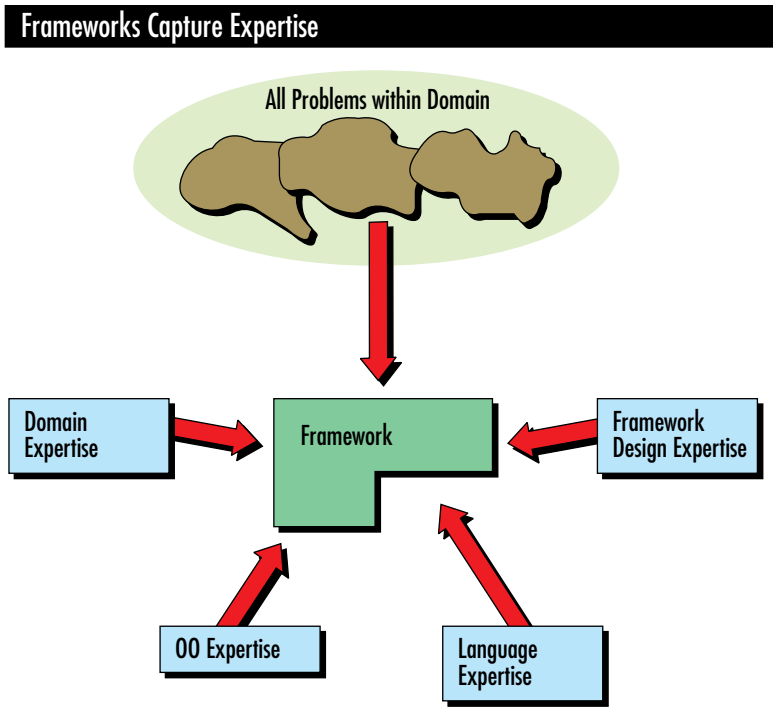


Figure 3. Frameworks capture expertise

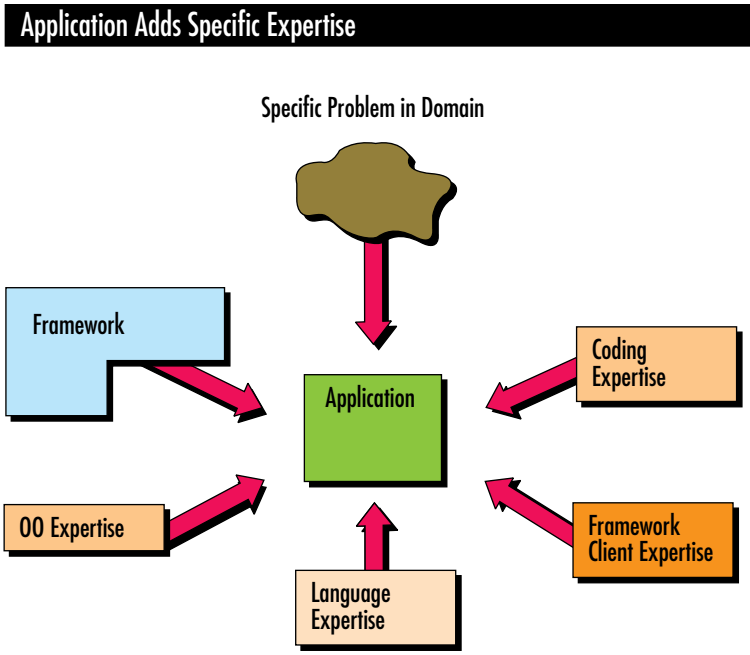


Figure 4. Application adds specific expertise

Working within the constraints imposed by the framework, you supply code to tailor the framework to solve your particular problem. In this manner, you turn the abstract solution represented by the framework into a concrete instance of an application, as shown in Figure 4.

Note that while you need to know how to use the framework in order to solve the problem, you do not need to be a domain expert in order to solve the problem. This is because, through use of the framework, you reuse the design captured by the framework. In effect, you inherit the domain expertise and problem-solving ability of the designer of the framework. A framework is a form of expert system.

Multiple Frameworks

So far, we have discussed applications and frameworks in terms of a one-to-one relationship: one application per framework, one framework per application. In practice, an application typically is implemented using multiple frameworks, shown in Figure 5.

When you require multiple frameworks for an application, the application consists of code for each framework.

Kinds of Framework Use

Frameworks are very flexible programming constructs, more so than procedural or class libraries. You can use frameworks in three different and complementary ways:

- **Use as is.** Use the framework as is, like a specialized class library. Some frameworks provide sufficient default behavior that they can be used out-of-the-box. (This is the simplest kind of use and does not preclude more sophisticated uses of the framework—it just means that the more sophisticated kinds of use are optional, rather than required.)
- **Complete**. Add code to the framework to implement specific capabilities. It is important to keep in mind that a framework represents a generic design solution, not any one solution. A framework does not have to exhibit complete default behavior—it may be only partially filled in. And a framework might not even be able to execute as delivered (it might be abstract, requiring developer-supplied code to make it concrete).
- **Customize**. Replace parts of the framework implementation. This is the most sophisticated and radical way to use a framework. You can

replace a little or a lot, or even the entire implementation. You can even implement some or all of the code in hardware (such as a graphics accelerator). In all cases the same interface to the framework is preserved, and programs that ask the framework for services do not need to know about the changes that you have made to the framework's underlying implementation.

Benefits of Frameworks

Frameworks return a number of benefits to developers. Some of these benefits are simply attributable to the fact that frameworks support design and code reuse; these benefits are also present when using well-designed class libraries, although to a lesser extent. Other benefits are unique to frameworks, and are advantages that frameworks have over both procedural and class libraries:

- **Less code to design and implement**. Much of the program's design and structure, as well as its code, already exists in the framework. By providing the infrastructure, the framework dramatically decreases the amount of standard code that you must design, code, test, and debug. You write only the code that extends or specifies the framework behavior to suit the program's requirements.

  Because the infrastructure of the framework is already in place, you write code only as required by the framework or to override some default behavior of the framework that is inappropriate for the application (this is sometimes called *programming by differences*). Typically, the amount of code required for an implementation is a small fraction of the code required to write the same application from scratch. This provides a corresponding decrease in the effort, time, and cost required to implement the functionality.

- **More focus on areas of expertise**. You can concentrate on your particular problem. You do not have to be an expert at writing user interfaces, networking, or printing to use the frameworks that provide those services. You can focus on your true value-added area. Just as standard programming interfaces insulate software routines from system dependencies and standard utilities facilitate development, frameworks provide standard solutions. This frees developers who are not necessarily experts in a certain area from the complexity of the underlying details. In this manner, frameworks create an environment in which solving domain problems—not programming problems—is possible. All you need to know (at least initially) is how to use the basic parts of the framework to get the job done.

Multiple Frameworks Used in Applications

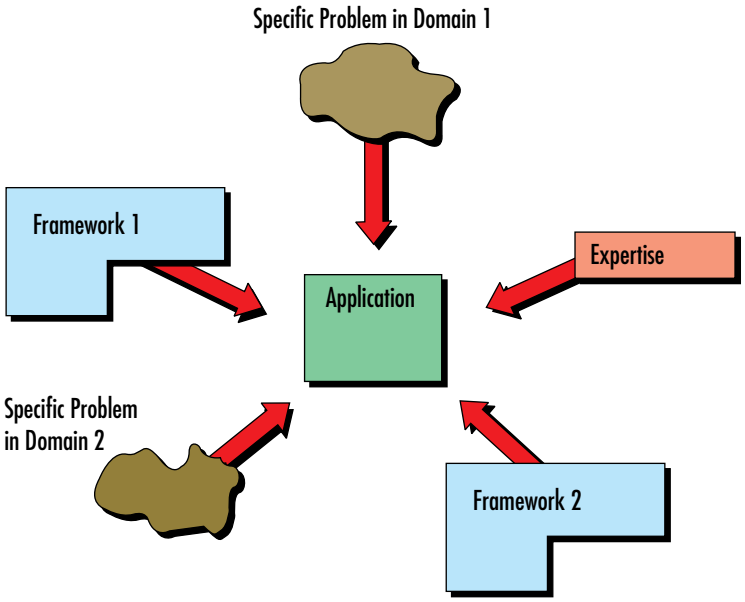


Figure 5. Applications can use multiple frameworks

- **Proliferation of expertise**. Good software design in a particular area requires domain knowledge that you typically acquire only by experience. Corporate and commercial development organizations as well as systems integrators have this acquired experience in particular areas, such as manufacturing, accounting, insurance, or financial applications. Frameworks allow organizations to package the common characteristics of their expertise.

  Frameworks and the embodied expertise behind the frameworks are a strategic asset for internal use and for those organizations that see business opportunities for reselling specialized knowledge.

  For example, frameworks give systems integration companies with expertise in vertical markets a distribution mechanism for packaging, reselling, and deploying their expertise.

- **More reliable and robust code**. Code reused through frameworks has already been tested and integrated with the rest of the framework (and with the system as a whole). This allows an organization to build from a base that has been proven to work in the past, and minimizes the amount of testing required.
- **Improved consistency**. Because frameworks embody expertise, you solve problems once—when first creating the framework—and you use the business rules and designs you capture in a framework consistently across all problems in the framework's domain.

  Additionally, frameworks enforce the relationships among the objects and classes in the framework, providing a higher degree of consistency than is obtained with either procedural or class libraries.

Frameworks provide a mechanism for reliably extending functionality.

- **Improved integration and interoperability**. Because all programs based on a particular framework reuse not only common code but also common design, developers who are working on similar programs have a better chance of understanding and working with each other's code, and the programs they develop are more likely to work together consistently and reliably. Because standard behaviors are captured in the framework, there is consistent behavior from product to product.

  Frameworks support a high degree of integration among multiple customizations. Much as individual objects hide their internal complexity and present a simplified interface for use by other objects, many different programs can use frameworks simultaneously in a way that allows the programs to share common behavior without interfering with each other's specialized implementations. This is possible because the client API of the framework remains unchanged.
- **Reduced maintenance overhead**. Fixing a bug once in a framework fixes the same bug in all the software derived from that framework. Similarly, when you add new features or capabilities to a framework, the additions automatically appear and work in all applications based on the framework. And because you make the changes in only one place, you minimize the chance of introducing errors in the code.

  Maintenance is far easier, because you amortize maintenance of a framework over many implementations. Because the implementation adds or changes only the things that are unique to the particular implementation, there is less new code.

  Also, quality of the final product is higher. Because you constantly reuse the framework, it becomes very robust. Thus a new product contains significant amounts of mature code in the framework, plus a smaller amount of new code in the implementation, resulting in higher overall quality.
- **Orderly program evolution**. Frameworks provide a mechanism for reliably extending functionality. While objects and class libraries provide interfaces for extending functionality at a fine-grained level, frameworks provide this flexibility at a higher level. In this manner, you can develop applications by using the framework as a starting point and writing smaller amounts of code to modify or extend the framework's behavior. You can add these extensions without sacrificing compatibility and interoperability because the interfaces are well-defined.
- **Enculturation**. Developers who are used to dealing with frameworks tend to think in terms of generic solutions rather than special solutions, with the result that you can often design your own programs as frameworks that you or others can reuse.

Taligent, along with many other leading industry players, foresees object technology as a way to change both the development and use of software. Taligent's approach, hosting the CommonPoint application system on existing 32-bit operating systems, enables users and developers to make the change incrementally, at a pace that preserves the value of current information technology investments and fosters new innovative solutions.

Software developers will be able to focus on what they do best: build solutions that address real-world business problems and issues. End users will become more task-focused, with intuitive applications that enable them to use the computer as a powerful tool in accomplishing their work. And groups of people will be able to effectively collaborate on business-critical projects and problem solving.

Taligent's vision and technology will provide a common base for applications, a common platform for heterogeneous computing, and a common environment for people to work together—in essence, a common point for a new generation of computing.

For more detailed information on system features and capabilities, look for two new books coming early this summer from Addison-Wesley:

- *Inside Taligent Technology* describes what Taligent technology can do, why it is important, and where the people who created it believe it is going. It is written by Sean Cotter, a freelance writer, and Mike Potel, vice president of Technology Development at Taligent.
- *The Power of Frameworks* illustrates the value of frameworks in application development. Written by Taligent, it includes a demo CD.

You can also browse the Taligent web page: http://www.taligent.com.

Joyce Uggla, Taligent, Inc., 10201 North De Anza Boulevard, Cupertino, CA 95014-2233. 1-800-288-5545. Ms. Uggla is staff technical editor at Taligent. She has a Master's degree in Education from Stanford University, and has been writing and editing technical information in the Silicon Valley for 15 years.

TECHNICAL SUPPORT FOR SOLUTION DEVELOPERS

Solution Developer Operations (SDO)—a recently announced unit at IBM—wants the latest level of solution developer's products to run at optimum performance on the latest IBM platforms, exploiting the power and capabilities of those platforms. And that requires support.

SDO offers technical support to all worldwide solution developers working on OS/2 and AIX to help them exploit IBM technology while developing on or porting to IBM platforms. The OS/2 Developer Assistance Program offers technical support to OS/2 developers, while the POWER Team offers technical support for AIX developers.

Mass communications provide a vital part of the technical support. IBM and solution developers are leveraging the possibilities inherent in Internet and commercial online services through white papers and documents on migrating solutions, code fixes, plus assistance with questions during development via forums and bulletin boards. Information and support via Internet and commercial services are available to any solution developer—no defined relationship with IBM is required.

SDO wants to go beyond the generalized support delivered via Internet and commercial services; they want defined relationships with worldwide solutions developers that will promote success for everyone. For AIX developers, the POWER Team provides programs that will accelerate developing on and porting to AIX while exploiting the latest technology, such as Symmetric Multiprocessing (SMP)

As a member of the POWER Team, you can use these services:

- Ask directed questions of experts who specialize in supporting developers
- Request customized assistance in determining how to exploit new IBM technology
- Participate in beta programs
- Use Developer's Discount Program to purchase systems for use in development

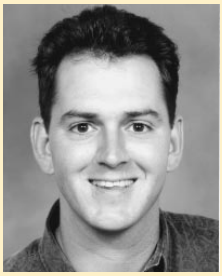
To join the POWER Team, call us at 1-800-222-2363. If you are considering developing on or porting to OS/2, SDO can help. Call 407-982-6408 for more information. (See "IBM Marketing and Technical Services" in this issue for more information about support for solution developers.)

—John Falvey, IBM Corporation

AIX Questions

Compiled by Brad Townley

The AIX Solution Provider Technical Support Group in Austin, Texas, supports software vendors who are developing or porting applications to AIX. This article is a compilation of questions that are frequently asked by vendors. The name of the responding Technical Support Group staff member appears after each response.


Brad Townley

How can I load OpenGL™ extensions in Common Desktop Environment (CDE) on AIX 4.1.2?

Add the extensions in the `/usr/lpp/X11/defaults/xserverrc` file. For example,

```
EXTENSIONS=-x mbx -x abx -x GLX.
```

—Brad Townley


Daryl Green

My modem supports speeds up to 38400 baud. Even though I set the TTY port for 38400, it always connects at 9600. How do I get it to connect at a faster speed?

Most Hayes®-compatible modems have a register that controls maximum line speed. The general rule of thumb is to set the TTY port to the fastest setting that your modem supports, then make sure that the value in that register is set to correspond to the speed at which you want to make a connection. See the owner's manual for your modem for two items: to locate the specific register number that will need to be set and to determine which register values correspond to which modem speeds.

—Daryl Green


Viral Shah

I use my RISC System/6000 (RS/6000) as a gateway. Since upgrading from AIX 3.2.5 to 4.1.1, it will not forward packets. How do I fix this?

One of the configurable options in the AIX kernel is `ipforwarding`, which specifies whether the kernel should forward packets. An `ipforwarding` value of 1 forwards packets when they are not for the local system, while a value of 0 prevents forwarding. In AIX 3.2.5, this value was set to 1 by default; in AIX 4.1, it is set to 0 by default. Although you can change this with the command `no -o ipforwarding=1`, you will also need to configure the `/etc/rc.net` file to set this value at boot time. This prevents it from being reset to the default value of 0.

—Daryl Green

Why does pinging a local machine take about 1.5 minutes when it should take only about 1 to 3 milliseconds?

If you have `/etc/resolv.conf` but you are not using domain name server, then delete the file. If you are using domain name server, make sure that the name of domain name server and domain are correct. Also ensure that `/etc/hosts` file does not contain any invalid entries.

—Viral Shah

I am having trouble logging in as a Network Information Service (NIS) user using Desktop login.

APAR IX47158 fixes this problem. You can also use the following workaround.

1. `cp /etc/rc.boot /etc/rc.boot.orig`

2. Comment out the DTRUN assignment line in /etc/rc.boot:

```
  #DTRUN=`/usr/bin/grep " dt:"
/etc/inittab | /usr/b
```

The next time you boot the machine, the graphical boot sequence will be disabled and <dtlogin> will be started after NIS.

—Viral Shah

When lslv is run on a logical volume, the Mirror Write Consistency field is always set to on. Does this mean that mirroring has been enabled even though the number of copies is just 1?

The Mirror Write Consistency field is part of the standard output of the lslv command. It defaults to ON, but this has meaning only when the number of copies is greater than 1. When the number of copies is 1, mirroring is not enabled. From a performance standpoint, mirroring is costly; mirroring with Write Verify is even more costly (extra disk rotation per write); and mirroring with both Write Verify and Mirror Write Consistency is the most costly of all (disk rotation plus a seek to Cylinder 0). Although an lslv command will usually show Mirror Write Consistency to be on for non-mirrored logical volumes, no actual processing occurs unless the copies value is greater than 1. Write Verify, on the other hand, defaults to OFF, since it does have meaning (and cost) for non-mirrored logical volumes.

—Priyamvada

Is it acceptable to have paging space of different sizes on several volumes?

All paging logical volumes should be approximately the same size. Paging spaces of different sizes may create problems. As the smaller ones become full, paging activity will no longer be spread evenly across all the physical volumes.

All processes started during the boot process are allocated paging space on the default paging space logical volume. After the additional paging space logical volumes are activated, paging space is allocated in a round-robin manner in 4 KB chunks. If you have paging space on multiple physical volumes and more than one paging space on one physical volume, you will no longer be spreading your paging activity over multiple physical volumes.

—Priyamvada

Priyamvada

Why are the special "box" characters not appearing on an N40 when I use smitty?

To make graphics characters (in this case, box characters), the font must be an IBM-850 font. Entering `aixterm -fn Rom11` will open a window with the correct font. Look in `/usr/lib/X11/fonts/fonts.dir` for a list of all the available fonts.

—Sai Ramanath

Why is the `VPATH` variable not being evaluated by `make` on AIX 3.2.5?

The `VPATH` usage for targets has been withdrawn from AIX 3.2.5 and later versions. Use `$?` instead of the prerequisite names, which forces `make` to look for them in the current directory.

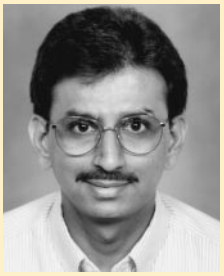
—Sai Ramanath

Sai Ramanath

Can I use the HeapView Debugger with a threaded program?

You cannot use HeapView Debugger with a threaded program. The HeapView Debugger is statically bound with `libc.a`, so a threaded program will not work as expected. It also does not do any interlocks; therefore, it runs the risk of corrupting its own memory.

—Darshan Patel



Darshan Patel

I am using the C Set++ compiler. Is there a flag that specifies the architecture on which the executable program will be run? (See POWER Notes in AIXpert, February 1995, page 65 for more information.)

Use the `-qarch=option` flag to specify the architecture on which the executable program will be run. Use the following suboptions to produce an object containing instructions that will run on the specific hardware platform:

- **-qarch=com**: All the POWER, POWER2, and PowerPC hardware platforms (default)
- **-qarch=pwr**: Any POWER hardware platform
- **-qarch=pwr2**: POWER2 hardware platform
- **-qarch=pwrx**: POWER2 hardware platform (same as `-qarch=pwr2`)
- **-qarch=ppc**: 32-bit PowerPC hardware platform

—Darshan Patel

Fred Arnold

How can I determine how much space an install requires?

In AIX 4.1 and later, one of the install options is to go through the install without actually installing AIX. This option displays the required space.

—Fred Arnold

How much available space is required in `/tmp` when installing or creating a `mksysb`?

The system should have at least 8 MB of available space in `/tmp`.

— Fred Arnold

How can I change the number of processes assigned per user?

Change the number of processes assigned per user (as root) as follows:

1. Enter <`smit/smitty`>.
2. Select System Environments.
3. Select Change/Show Characteristics of Operating System.
4. Select Maximum number of processes allowed per user.

This can also be done from the command line where `n` is a number between 40 and 10,000:

```
chdev -E -l sys0 -a maxuproc=n
```

—Jeff Simon

How can I build shared libraries with C++ code? How do I ensure that the static constructors are called? I am building C shared libs with `ld` and do not know if I need to use XLC or some special `ld` option.

To ensure that the static constructors in a shared file are properly initialized, construct a library

with shared file information by using `/usr/lpp/xlC/bin/makeC++SharedLib`. This will create a C++ shared library from an export list and a list of object files and ensure that static constructors are properly initialized. The output of this program is a file containing the shared part of the library, constructed so that initialization will be done correctly. This file can then be placed into an archive library using the `ar` command.

The steps to use `makeC++SharedLib` are documented in the *C++ User's Guide*:

- *C Set++ User's Guide Version 2.1* (SC09-1605, pages 247—251)
- *XLC++ User's Guide Version 1.1.1* (SC09-1472, pages 144—146)

If there are static constructors or a reference to a shared library with constructors (such as `iostreams`), you must link with XLC for everything to be initialized correctly. Linking with XLC automatically brings the necessary configuration and library files needed (that is, `munch`, `/usr/lpp/xlC/lib/crt0.o`, and `/usr/lpp/xlC/lib/gcrt0.o`). To see which flags are automatically brought in by XLC, look at `/etc/xlC.cfg`.

—Jeff Simon

How can I change the logo for CDE in AIX 4.1.2?

Set the Dtlogin*logo*bitmapFile in the `/usr/dt/config/C/Xresources` file to Dtlogin*logo*bitmapFile: `/usr/local/dt/bitmaps/Mylogo.bm`.. This can be a bitmap or pixmap.

—Brad Townley

Brad Townley, Internet: bjt@austin.ibm.com. Mr. Townley is a project manager specializing in graphics. He has a BA in Economics from the University of Illinois.

Jeff Simon

Open Software Foundation Certifies AIX/DCE

The Open Software Foundation (OSF) named IBM's AIX Distributed Computing Environment (AIX/DCE) as the industry's first certified implementation.

The certification process assures customers that any implementation of DCE—a base for open, distributed, client/server computing—fully interoperates with other certified DCE solutions.

"IBM's fundamental goal is open, distributed client/server computing across multiple systems. DCE provides a widely used infrastructure across multiple systems, and OSF certification of IBM's AIX/DCE assures interoperability," said Don Haile, general manager, IBM Networking Software Division.

DCE provides a comprehensive, integrated set of services that support transparent communications and resource sharing across distributed, multiplatform networks. Customers can design and implement secure applications with DCE's directory, time and security services.

With nearly a quarter of the 100 largest U.S. corporations using IBM DCE, the value of DCE technology is clearly recognized. IBM customers are using DCE to protect their investment in hardware, software, and application development costs.

IBM provides DCE implementations on MVS™ Open Edition, OS/400®, AIX, OS/2, and Windows 3.1 platforms. The comprehensive set of fundamental services in DCE provide the same "look and feel" regardless of the underlying operating platform.

In addition to having the first OSF DCE certification, IBM in 1992 was the first supplier to provide a commercially available DCE offering. IBM followed with the industry's first DCE-based distributed OLTP technology, and the first PC network operating system based on DCE.