OBJECT ORIENTATION: FROM APPLICATIONS TO ENVIRONMENTS

by
Ronald F.E. Weissman, Director of Higher Education
Kristofer A. Younger, Senior Technical Consultant
NeXT Computer, Inc.

FROM OBJECT-ORIENTED PROGRAMMING TO OBJECT-ORIENTED APPLICATIONS

Despite sudden industry attention, the object-oriented software revolution is proceeding slowly. Though the underlying model dates back to research at Xerox Parc in the 1970s, it is only recently that object-orientation has become a mainstream paradigm. As we enter the 1990s we are seeing large-scale adoption of the 'object-oriented' model, even in the more retrograde sectors of the computer industry.

In the most conceptually advanced segments of the industry, whole software systems, such as the entire NeXT development environment, depend on object-orientation as the primary means for software engineering. The ability of object-oriented languages to develop modular components and create reusable code has made them the 1990s solution to productivity and software maintainability. The current focus of object-oriented technology has been on improving the productivity of programmers, which is too limited. We believe the true benefits of object-orientation will accrue to end-users of software applications as much as to professional programmers.

What is object-orientation from a user's perspective?

At its most elemental level, object-oriented systems revolve around two key concepts: *objects* and *messages*. All software in these systems is constructed of objects that can send and receive messages. An object-oriented user-interface system, such as NeXTstep, is designed around its objectsÐbuttons, cursors, windows, scroll bars, and menus. Objects communicate with other objects by sending messages. This is the only way objects actually do anything. Lines of text change position on screen, for example, when a scroll bar object sends a message to the text view object to move the lines appropriately, as specified by the user.

To use an object, one need only know the messages that the object understands (the object's *protocol* or *interface*), not how it works internally. To use an object-oriented spelling checker, for example, one need only know that the spelling checker is invoked by sending it a "checkSpellingOfParagraph:" message. One need not know how, exactly, the code works or what kinds of search, sort, and comparison algorithms give the spelling checker its power. Indeed, as long as the messages sent to the spelling checker remain the same, one could replace today's spelling checker with a more powerful version in a manner absolutely transparent to users.

We are accustomed to view a well-designed software application as being built from a set of objects. A spreadsheet, for instance, is composed of cell objects and matrix objects. But what if we viewed commercial software applications themselves as objects? What if tomorrow's end-user worked in a world in which spreadsheets, databases, word processors, statistical tools, presentation packages, graphics packages, and communications tools were not standalone 'solutions' but software objects? Then, tomorrow's software environment would be composed of applications that could send and receive messages from other applications. ThisÐthe real promise of object-orientationÐwould give hundreds of thousands of serious computer users the ability to create rich, customized environments, and not simply empower a few thousand professional programmers.

What will object-oriented applications be able to do?
First, they will be able to send, receive, and share data with other applications. A bibliography built in this way would, for example, be able to provide citation data to research databases as well as provide footnote data to word processed papers and articles. Second, object-oriented applications will be able to provide services to other applications. An object-oriented spelling checker might offer its spell-checking services to any application using text, whether it was a spreadsheet cell, a database record, or a word processed document.

Release 2.0 of the NeXT system software supports this notion of a 'Service' that one object-oriented application can provide to another application. Through the Services menu, one application can message another and ask it to perform a service. Envision, then, a new type of interactive computer environment: During the 1990s object technology will empower users through Object-Oriented Applications. Applications will become richer given their new-found ability to communicate with other applications. Today, the most advanced environments are composed of interacting object-applications based on the best of today's personal productivity applications. This environment is enriched by networked data sharing, support for messages, and support for an 'interpersonal' style of computingÐallowing groups of people to share data and tools collaboratively.

THE NeXT OBJECT ARCHITECTURE

How does NeXT's System 2.0 support object-orientation today?
Much has been written about NeXT's object-oriented programming tools, its Application Kit, and Interface Builder. What few realize, however, is how thoroughgoing this object orientation is. NeXT technology provides much more than an object-oriented language compiler. For the professional programmer, NeXT provides the common objects that most applications need. This allows NeXT to provide developers a full set of user interface objects including windows, scrollers, simple button-oriented cause-effect triggers, and a text object that would stand out as a great word processor on most other UNIX systems. From the Mach operating system kernel, through its user-level services, the NeXT software environment is fully object-oriented, enabling programmers today and end-users tomorrow to reap the benefits of a true object architecture.

The importance of Mach
At its most fundamental level, the NeXT environment is based on the architecture of an object and message system. NeXT chose Carnegie-Mellon University's Mach as its operating system precisely because Mach is built around an efficient object messaging and interprocess communication

architecture.

Mach's messaging facility is implemented by two kernel abstractions: *ports* and *messages*. A *port* is a protected communication channel to which messages may be sent and queued. The port is also the basic object reference mechanism in Mach; its use is similar to that of object references in an object-oriented system. That is, operations on objects are requested by sending messages to and from the ports that represent them.

Message passing is the primary means of communication both among tasks and between tasks and the operating system kernel itself. System services for example, are invoked by one task sending a message to another task that provides the desired service.

How does NeXT support object-oriented applications?
NeXTstep 2.0 provides three fundamental facilities for creating and manipulating object-oriented applications: the Speaker/Listener protocol, Interface Builder Palettes, and Workspace Manager Services.

1. Speaker/Listener
On top of Mach's interprocess communications facility, NeXT has built an interapplication communications facility. Each application on NeXT can have two communications objects: a *Speaker* object, which is used to send messages to other applications, and the Workspace Manager, and a *Listener* object, which receives messages from other applications. Both of these objects are built on top of Mach's ports. For example, if a user needs to define a word using the WriteNow word processor, the Speaker object in WriteNow sends the Listener object in Webster a "defineWord:" message. All message passing between applications is implemented via basic Mach messaging.

2. Interface Builder and support for object-oriented software
Interface Builder has increased its functionality in release 2.0 by dynamically supporting loadable groups of reusable objects *palettes*. A palette is a collection of objects ranging from such basic items as text fields, buttons, and sliders, to more complex objects which can represent databases or specialized charts that one drags off a palette window to drop into one's application's windows. In release 2.0, one can add new palettes (from commercial developers or one's own projects) of objects to Interface Builder's palette window and drag these new objects off into one's applications.

Objects from palettes may be examined via inspectors, so one can change the internal properties and attributes or default values from within Interface Builder. In release 2.0, Interface Builder knows how to generate project files (such as ".palette" files) of different kinds. Once a palette has been created, it can be shared with colleagues.

Release 2.0 offers users the opportunity to generate a proliferation of different kinds of small, highly functional palettes: visualization objects, control objects, music objects, graphic objects, and analysis objects. Many of tomorrow's 'paletted' objects will be full featured mini-applications. By dragging objects off palettes and linking objects, end-users with some programming experience will be able to create their own custom applications. In tomorrow's object-oriented world, users will be able to link database access objects (which will appear in late 1991 via NeXT's database object kit) to visualization objects, to create custom analysis toolsÐall without significant programming.

[David/Kris: We   need a screen shot of objects and palettes doing something useful. Perhaps a dbKit screen shot?]

3. Services
NeXTstep 2.0 introduced the concept of *user-level services*, which allows one application to take advantage of the tools provided by another application. Services are registered with the NeXT Workspace and become available to any application via the now-standard Services menu. Every NeXT computer includes, among others, these common services:

- Any application can message Digital Webster to define selected words.
- Any application can message Digital Librarian to search for a selected word or words in a user-specified body of reference or user-written text materials.
- Any application can avail itself of full communications services to mail, print, or fax any document.

In addition, a user can acquire commercial applications providing these kinds of services to other applications:

- HSD's *OCR Servant* accepts a ".tiff" files (bitmapped scanned images) and converts them into editable ASCII text; and
- *Presentation Builder* sends a matrix of numerical data (tab-delimited at one line per row) to the Lotus *Improv* charting application, *PBuilder*.

[David/Kris: We   need a screen shot of one app using the services of anther. Perhaps the HSD OCR stuff?]

3. FROM APPLICATIONS TO ENVIRONMENTS

In today's traditional applications environment, commercial and academic software developers typically view their products as standalone tools. Underlying platform technology, such as that provided by NeXT, offers the possibility of moving from a world of discrete applications to a world of cooperating, integrated, and user-customized environments. In higher education, for example, one can envision scholarly environments composed of component software application objects such as:

- word-processor, text-database, hypertext linking, concordance, and dictionary objects that communicate to create a humanist's scholarly environment.

- cartographic, statistical analysis, spreadsheet, and database objects that communicate to create a social scientist's environment.

- spreadsheet, simulation, database, and linear modeling objects that communicate to create an economist's environment.

- CAD, spreadsheet, project-management, instrumentation and data acquisition tools, specialized analysis tools, visualization, and statistical objects that communicate to create an engineering environment.

- MIDI tools, waveform editors, sampling and synthesis tools, and score editing objects that communicate to create a musician's environment.

The underlying technologies to create environments exist today. Through the Services registration process, applications can avail themselves of the functionality provided by other applications. And through dynamically loadable palettes of objects, tools can be combined into 'meta-applications' based on the cooperation of several smaller, functionally specialized tools.

The task of moving from applications to environments is one of vision, not one of technology. Commercial software vendors need to to understand the benefits of object technology as perceived by users; software developers need to embed an application's functionality in communicating objects, and make a rich set of their application's functionality available from within the Services facility.

By creating a new software paradigm and a world in which communicating objects come to replace today's world of standalone applications, we will, finally, have harnessed the vast potential of object-orientation to empower of tomorrow's serious computer users.