An Overview of Distributed Objects

The various components of OPENSTEP Enterprise along with NeXT's OPENSTEP for Mach product all contain the same version of NeXT's Foundation technology. An early version of the Foundation framework, without the new Distributed Objects (DO) functionality, was shipped with version 1.1 of the Enterprise Objects Framework.

Briefly, for reference, some of the newer features of the Foundation framework are:

- An abstraction of file system interfaces (NSFileManager)
- Abstractions of file descriptors and pipes (NSFileHandle, NSPipe)
- Abstractions of child and background processes (NSTask)
- A new Distributed Objects subsystem

Compatibility

The existing Distributed Objects subsystem (also referred to as "NXDO"; the new Distributed Objects subsystem is referred to as "NSDO") continues to be supported for both compatibility and migration needs. Both subsystems exist and can run simultaneously. Both share the same underlying Mach IPC services (name look-up and transport). The two DO subsystems do not interoperate, however, in that proxies from one subsystem cannot be used in the other. This is primarily due to the new NSObject family of objects and its new memory allocation strategy. Although the two subsystems don't interoperate, NXConnections can be run in the new environment.

Architectural Components

Distributed Objects has been implemented as a set of reusable components. NSDistantObject proxies capture messages intended for delivery elsewhere, NSConnections manage exported and imported objects, an NSPortCoder transcribes messages (and objects) using the same NSCoding protocol that is also used for archiving, and an NSPort actually packages and sends bits across the "wire."

Memory Reclamation

The new Distributed Objects utilizes Foundation's NSAutoreleasePool mechanism to

aggressively recover all of its resources. This allows an exported object to be immediately (and transparently) recovered as soon as all of its clients relinquish their proxies. Also, a server can construct and return a custom object to a client without an unrecoverable memory situation as exists in NXDO.

Proxies and Subclassing

The Objective-C forwarding mechanism has been augmented by the NSInvocation class. NSInvocation captures, in object-fashion, the call frame that's not directly supported by a target class. The invocation object allows you to reset parameters and targets, and capture and return result values of arbitrary type (such as **struct**s and **double**s). The general notion of a proxy is encapsulated in the NSProxy class, while the specific notion of a Distributed Object is captured by a further subclassable NSDistantObject.

If there is constant information about an object for which you want to avoid sending messages remotely, you can subclass NSDistantObject. In this case, you can add appropriate instance variables and use this special subclass instance to replace the object via NSObject's **replacementObjectForPortCoder:** method. On the remote side, the accessor methods for this constant information are implemented to access the custom subclass instance variables directly.

Another reason to subclass NSDistantObject is if the distant object responds to methods that cannot be encoded with Distributed Objects—perhaps it returns a union, or an array of integers. In this case, the custom subclass should implement these methods and use private alternative methods that can be sent via NSDO.

NSCoding and the New "byref" Keyword

A common protocol is used both for distributed objects and for archiving. This allows a class to implement the encoding and decoding routines only once rather than (essentially) twice as in the NXDO subsystem. A new Objective-C keyword, **byref**, has been added to the language. It is the complement to the existing **bycopy** keyword and indicates that a proxy reference should constructed for this object. The Foundation collection and value objects (NSArray, NSDictionary, NSValue, and so on) now copy themselves by default rather than sending proxies as was the case in NXDO. The **in**, **out**, **inout**, **oneway**, **bycopy**, and **byref** keywords may now be used in interfaces and categories directly, instead of having to be contained within the **@protocol** construction.

A class can ask that another class be used when decoding instances (using NSObject's **classForPortCoder** method). As an example of this technique, NeXT's private NSString subclasses do to decode themselves as NSStrings, so that the private NSString subclasses don't need to be present everywhere strings might be used.

NSConnections and New Delegate Methods

NSConnections use the NSRunLoop to gather new input requests and responses. NSRunLoops are bound, one-to-one, to threads. Each runloop has a set of inputs (NSPorts, file descriptors, and timers) that can be serviced in one or more "modes." NSConnection lets you augment or restrict the modes in which to be run in addition to the default mode (for example, the Application Kit has a modal panel mode that is of some interest to OpenStep programmers). Use of the runloop mechanism allows a client or a server to detect the death of its correspondent without needing to resort to timeouts. For more information on runloops, see the NSRunLoop class specification in the *Foundation Framework Reference*.

One default connection per thread is used to share common configuration information such as timeouts, delegates, and the shared root object. (Note: default timeouts are now infinite, instead of NXDO's 15 seconds). A notification is sent whenever a new connection is formed so as to allow configuration refinement. The connection object also collects and provides usage information in a statistics dictionary.

A client or server may wish to block processing of unrelated DO activity during processing of a DO request. This can be enabled by issuing **setIndependentConversationQueueing:**YES. This is useful if a client or server temporarily changes its internal state and cannot logically process new requests until the existing one is resolved.

Multi-threaded operation is now fully supported, whereas in NXDO there were restrictions about the type of callbacks that could occur. Connections normally operate in only one thread, however, to avoid subtle problems of accidentally vending non-thread safe objects. NSConnection's new **enableMultipleThreads** method removes this restriction, as does its new **addRunLoop:** method.

Other new features have been added through the use of optional delegate methods:

- Authentication. Every DO message has an optional section for authentication data. This can be used asymmetrically so that only clients are required to authenticate incoming server messages. The delegate could also choose to strongly authenticate only the first message. We currently do not supply an authentication agent.

- Encryption. Analogous to authentication.

- Asynchronous request handling. A delegate method, if implemented, is provided with a NSDistantObjectRequest object that contains the invocation that would be sent. The delegate can choose to either process this invocation in any manner it chooses or simply have the connection proceed with normal processing. So, for example, a

delegate might enqueue the request and have it implemented by a pool of threads. The request implements a method that will return its result to the original requestor.

- Conversation queueing. The marker used to identify conversations across multiple address spaces can be set by the **createConversationForConnection:** delegate method. It can be examined by the conversation method in the NSDistantObjectRequest object or by NSConnection's **currentConversation** class method.

Alternate NSPort Transports

The lowest layer of the Distributed Objects subsystem can be substituted with a custom NSPort implementation. An NSPort subclass need implement only a few methods for scheduling and sending byte streams that have already been encoded to represent Distributed Object requests, replies, and other administrative messages. An example illustrating usage of raw TCP sockets will be included on all platforms.

For a general introduction to NeXT's Distributed Objects system, see **IntroDistObjects.pdf** in **NextLibrary/Documentation/NextDev/Old_PDO_Reference/DistributedObjects** (relative to the directory in which you installed PDO). For up-to-date information on specific Foundation classes, see the *Foundation Framework Reference* in **NextLibrary/Documentation/NextDev/Reference/Foundation**.