IONA

Technologies

Making Software Work Together™

OrbixWeb 3.2 White Paper

IONA Technologies
August 1999

Making Software Work Together™

M2366

Making Software Work Together™

Summary

This white paper introduces OrbixWeb 3.2, from IONA Technologies. OrbixWeb is a standards-based environment for developing and deploying scalable and secure Internet systems. This white paper starts by introducing the OrbixWeb architecture and the benefits of using OrbixWeb. It then describes briefly how each component of OrbixWeb works and provides an overview of the new features in version 3.2.

IONA Technologies

Making Software Work Together™

Contents

Introduction

The problem of integrating incompatible software systems is one of the largest challenges that organizations face today. Fully integrated enterprise systems must now also extend to the Internet. Developing and deploying Internet-enabled systems imposes a specific set of requirements, which all Internet solutions must address.

Integrated Internet applications must be robust, reliable and secure. Increasingly, the Internet is used in the infrastructure of mission-critical systems. For example, Internet banking, online shopping, and business-to-business sales all require a reliable and secure infrastructure.

Internet systems must be scalable, flexible, and capable of accommodating a huge range of applications, from existing legacy applications to third-party packaged software. For example, a Java applet or an ActiveX control may need to communicate with a UNIX server or an OS/390 mainframe.

Businesses must be able to react quickly to take advantage of new market and technology opportunities— and the systems they employ must be dynamic to adapt just as quickly. Mergers and acquisitions, internal reorganizations and changing business processes increase the importance of this requirement.

This potential for software diversity and dynamic business environments require a reliable, flexible, and dynamic solution to the problem of software integration. Middleware is the fastest growing area of technological advancement in this area. Like Java, middleware is platform independent. However, middleware also brings the added benefit of language independence. Middleware solves the problem of making diverse software systems work together across networks of any scale, leaving you free to choose the best applications for your needs.

The Object Management Group (OMG) Common Object Request Broker Architecture (CORBA) provides the foundation for an open, flexible, object-oriented approach to software integration. CORBA is the widely adopted and deployed standard for middleware. OrbixWeb builds on CORBA to provide the complete Internet solution to the integration problem.

The OrbixWeb Architecture

OrbixWeb is a software environment that allows you to build and integrate enterprise systems on the Internet. OrbixWeb supplies the development tools that enable you to build secure Internet applications; from thin clients to enterprise servers. OrbixWeb also supplies the deployment environment that enables you to deploy and configure enterprise Internet systems.

OrbixWeb 3.2 Professional Edition consists of five components:

- *The OrbixWeb Object Request Broker (ORB)*. The OrbixWeb Java ORB provides the infrastructure that lets objects communicate directly, independent of network location, operating systems, and programming languages. The OrbixWeb 3.2 ORB implements the OMG CORBA specification, version 2.1, and revision 1.1 of the OMG IDL/Java Mapping.

- *Orbix Wonderwall*. Orbix Wonderwall is an Internet firewall that provides the security required for Internet CORBA systems. Orbix Wonderwall provides the firewall technology to filter messages sent to CORBA objects.

- *The Orbix Code Generation Toolkit*. The Code Generation Toolkit provides a set of rapid application development tools for CORBA applications. The Code Generation Toolkit enables you to create scripts that generate Java code from an IDL file.

- *OrbixNames*. OrbixNames is IONA's implementation of the CORBA Naming Service. This allows clients to locate the objects they require. OrbixNames supports large-scale systems by enabling you to balance the load of client requests across multiple servers.

- *OrbixCOMet*. OrbixCOMet provides a two-way bridge between the worlds of the Microsoft Component Object Model (COM) and CORBA. Using OrbixCOMet, COM applications can communicate with CORBA objects and CORBA applications can communicate with COM objects. OrbixCOMet transparently converts messages from one object model to the other.

OrbixWeb 3.2 Standard Edition includes the OrbixWeb ORB and OrbixNames components.

Figure 1 on page 3 shows how OrbixWeb components work together to integrate applications in an example OrbixWeb system. Some hosts in this system provide services by running applications that contain distributed objects. These applications can be new or old—wrapping a legacy application with distributed objects is a straightforward task.

Making Software Work Together™

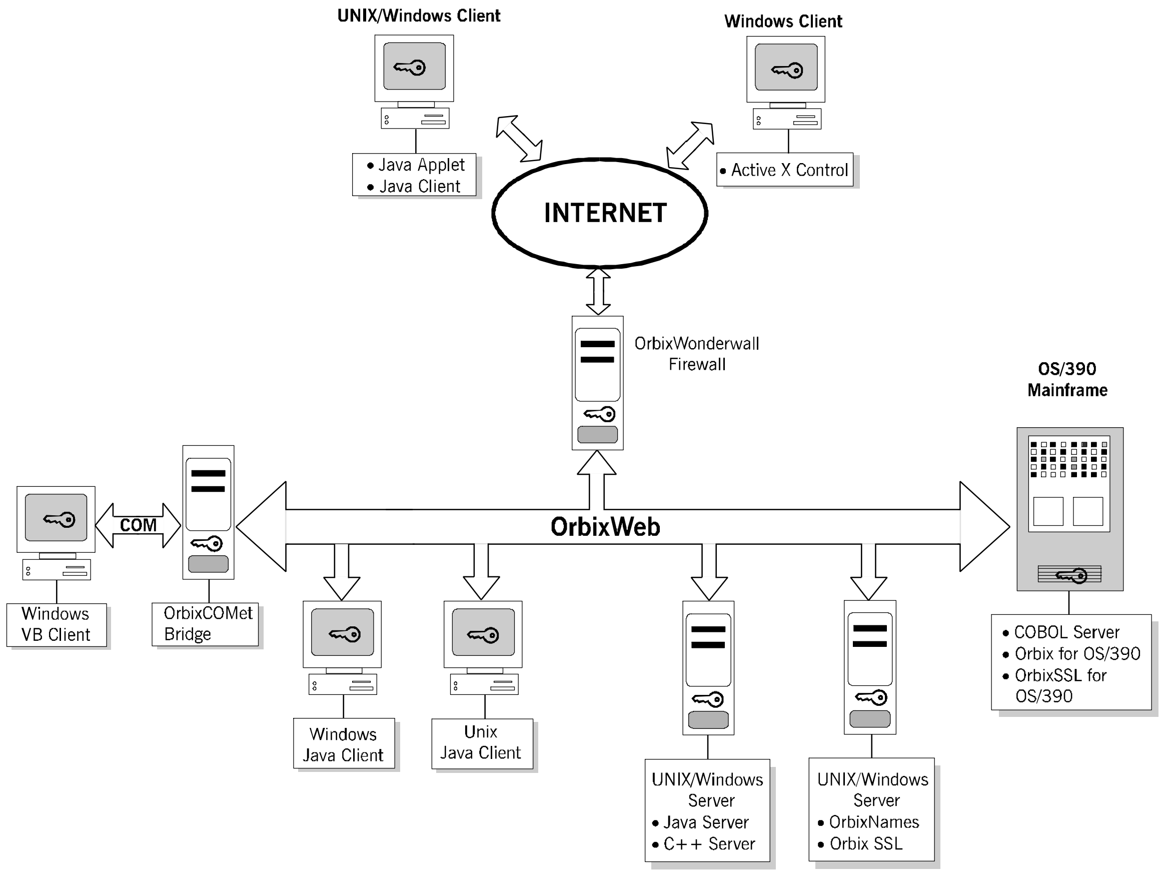


Figure 1. Components of an OrbixWeb System

Other hosts run applications that request the services of your distributed objects. These applications can be deployed on the Internet or within an intranet. For example, Orbix Wonderwall protects your network security by filtering the messages sent to your CORBA objects.

OrbixWeb acts as the underlying infrastructure, enabling your applications to communicate directly, regardless of their platform, programming language or location on the network.

Making Software Work Together™

OrbixWeb Applications

The pure-Java ORB is at the heart of the OrbixWeb architecture. An ORB is a software component that mediates the transfer of messages from a program to an object located on a remote machine. The role of the ORB is to hide the underlying complexity of network communications from the programmer.

An ORB allows you to create standard software objects whose member methods can be invoked by *client* programs located on the Internet or in your intranet. A program that contains instances of CORBA objects is known as a *server*.

When a client invokes a member method on a CORBA object, the ORB intercepts the method call. As shown in Figure 2 on page 4, the ORB redirects the call across the Internet to the target object. The ORB then collects results from the method call and returns them to the client.

CORBA specifies the mechanisms you can use to create and communicate with Internet objects in this way. Significantly, it also specifies the on-the-wire protocol that the ORB components must use when communicating with each other. This is the CORBA Internet Inter-ORB Protocol (IIOP), which runs over TCP/IP. This protocol is designed to provide efficient interoperability between ORB applications.

This common, standard protocol enables different ORBs to interoperate seamlessly. Using IIOP, a client developed using OrbixWeb can communicate directly with an object implemented with another ORB.

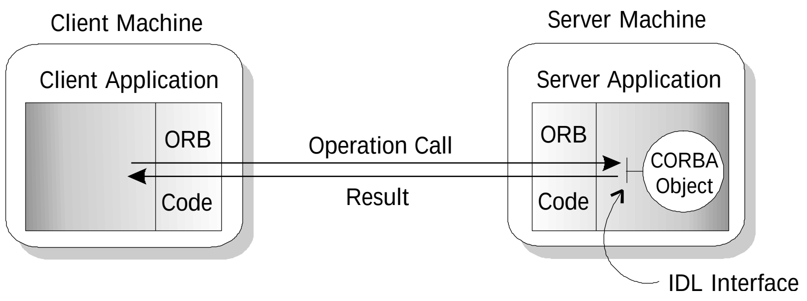


Figure 2. An ORB Mediates Calls from a Client to an Object

How CORBA Objects Work

CORBA objects are standard software objects implemented in any supported programming language. CORBA supports several languages, including Java, C++, PL/1, and COBOL. The objects can run on any platform supported by CORBA, including many UNIX platforms, Windows, OS/390, and AS/400.

With a few calls to an ORB application-programming interface (API), you can make CORBA objects available to client programs. Clients can be written in any supported programming language and can call the member methods of a CORBA object using the normal programming language syntax.

Although CORBA objects are implemented using regular programming languages, each CORBA object has a clearly defined interface, specified in the CORBA Interface Definition Language (IDL). The interface definition specifies which member methods, known as IDL *operations*, are available to a client, without making any assumptions about the implementation of the object.

To call operations on a CORBA object, a client needs only the object's IDL definition. The client does not need to know details such as the programming language used to implement the object, the location of the object in the network, or the operating system on which the object runs.

The separation between an object's interface and its implementation has several advantages. For example, it allows you to change the programming language in which an object is implemented without changing clients that access the object. It also allows you to make existing objects available across a network.

Developing OrbixWeb Applications

Consider a Java application in which a client application communicates with a CORBA object. To develop this application using OrbixWeb, you must:

1. Define the interface to your object, using IDL.
2. Compile your IDL definitions, using the OrbixWeb IDL compiler. The IDL compiler produces two types of Java code from your definitions: *stub code* and *skeleton code*. Stub code allows clients to call operations on a CORBA object. Skeleton code allows a server to create Java objects that implement IDL interfaces.
3. Develop your server application. The server should include a class that implements your IDL interface. When the server creates an object of this class, that object becomes available to clients through the corresponding interface.

4. Develop your client application or applet. The client locates a CORBA object created by the server and calls operations on that object, using normal Java method calls.

5. Compile and run your applications. The server should be compiled with the skeleton code generated by the OrbixWeb IDL compiler. The client should be compiled with the generated stub code.

Figure 3 illustrates this basic development process. The Orbix Code Generation Toolkit allows you to automate many of the coding tasks in this process. OrbixCOMet allows you to extend application development to include COM clients and servers.

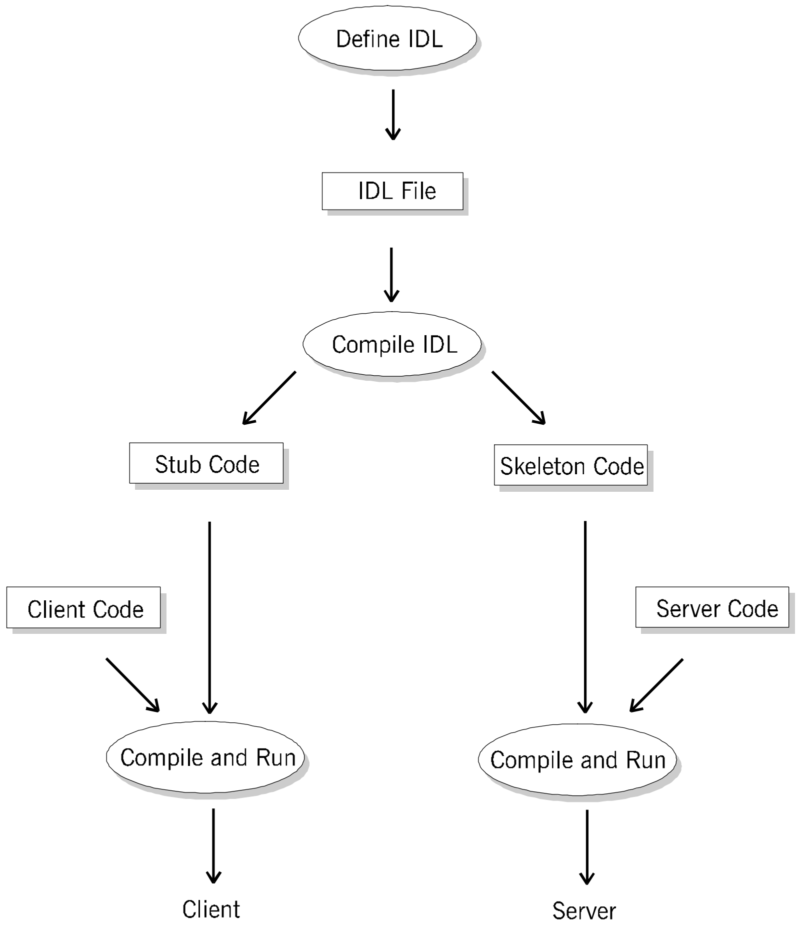


Figure 3. Developing an OrbixWeb Application

The Benefits of Using OrbixWeb

CORBA is a well-established and widely adopted standard for distributed programming. The wide application of CORBA means that you can integrate the broadest possible range of systems with a single, easy-to-use programming model.

CORBA also brings a standardized object-oriented approach to distributed programming. The many advantages to this include:

- *High component reuse*. CORBA objects, with their clearly defined interfaces, can be reused in many contexts.
- *Easy maintenance*. Because of the separation of interface and implementation in CORBA objects, you can update the implementation of an object at any time, without affecting the rest of your system.
- *High portability of components*. CORBA standardizes many of the coding requirements for objects, so you can transfer code from one platform to another with minimum effort.
- *Easy integration of existing systems*. Using IDL, existing applications can be wrapped in standard CORBA interfaces, allowing them to interact freely in your distributed system. These applications can run on any of the wide range of operating systems supported by CORBA.
- *Flexible integration of future systems*. CORBA programming is independent of platform, language or network choice. This open framework can easily accommodate the introduction of future systems to your enterprise.
- *Internet Interoperability.* The Internet Inter-ORB Protocol (IIOP) defined by CORBA ensures that your applications will interoperate with applications built on ORBs supplied by other vendors.

OrbixWeb is a proven, scalable, high-performance Internet CORBA implementation. However, OrbixWeb is not just an Internet ORB. The integrated OrbixWeb environment also provides the following benefits:

- *Rapid application development*
- *Internet security*
- *Load balancing*
- *Open interaction with COM*
- *Deployment support tools*

The remainder of this section describes how OrbixWeb delivers these benefits.

Rapid Application Development

OrbixWeb is designed to save development effort. OrbixWeb makes diverse software systems work together, enabling you to avoid extensive re-engineering or customized, proprietary systems. To further improve your productivity when integrating applications, OrbixWeb now includes the Code Generation Toolkit, a set of rapid application development tools for CORBA.

The Code Generation Toolkit includes a set of code generation scripts. These scripts take as input any IDL file, and generate Java or C++ code based on the contents of the file. For example, the toolkit includes a script that generates a complete sample application for a set of IDL definitions.

The toolkit enables you to get a simple application running, based on your own IDL definitions, with a minimal amount of coding. It also enables you to propagate changes in your IDL to application code very easily. The toolkit is fully extensible and allows you to add further code generation scripts, customized to meet the requirements of your system.

Internet Security

Internet client applications often need to communicate with CORBA objects in a secure manner. For example, if you are using OrbixWeb or OrbixCOMet to deploy Java applets or ActiveX controls that communicate with your CORBA objects across the Internet. OrbixWeb provides Internet firewall security through Orbix Wonderwall, IONA's Internet firewall. OrbixWeb provides Secure Sockets Layer (SSL) security through seamless support for OrbixSSL, IONA's implementation of the SSL version 3.0 standard.

Firewall Security with Orbix Wonderwall

When clients communicate with CORBA objects across security domains, system administrators must protect servers from corruption or malicious attack. The standard method for protecting servers is to use a firewall. Typically, a firewall forces all communications of a certain type through a particular port on a bastion host. It then filters and logs all messages before passing them to the main server.

If your applications can communicate with your server using IIOP, you can guarantee security only by filtering IIOP messages. Orbix Wonderwall enables you to do this. Orbix Wonderwall enables a system administrator to configure which elements of a distributed application should be exposed to the Internet. It then examines each incoming IIOP message and forwards only safe requests to your CORBA objects.

Orbix Wonderwall includes a full HTTP 1.1 implementation that allows Java applets to access CORBA objects, even from behind a client's corporate firewall. This technology allows the applet's ORB classes to wrap IIOP requests destined for Orbix Wonderwall in HTTP, preventing the client-side firewall from rejecting them.

SSL Security with OrbixSSL

OrbixWeb, OrbixNames, Orbix Wonderwall, and OrbixCOMet are now fully integrated with OrbixSSL. OrbixSSL provides Java and C++ integration with the SSL version 3.0 standard. SSL version 3.0 provides the following features:

- Authentication— providing protection against impostors. Based on X.509 certificates.
- Privacy— ensuring no data snooping through use of full or export strength encryption algorithms.
- Integrity— providing protection against data tampering using digital signatures.

OrbixSSL for Java provides full SSL version 3.0 capability through a set of downloadable, portable pure-Java classes. This supports 128-bit Data Encryption Standard (DES) and RC4 encryption. The OrbixSSL 3.0 product is available as part of the OrbixOTM 3.0 product suite.

Load Balancing

OrbixNames provides the mechanism by which clients locate objects in distributed systems. This makes OrbixNames central to the operation of your applications. As such, OrbixNames is a component that can coordinate the interaction between clients and objects.

OrbixNames uses its role as a coordinator to provide a simple, but powerful load-balancing system for distributed objects. OrbixNames allows you to set up groups of objects that can provide particular services to a client. When a client requests an object from a group, OrbixNames selects an object using a random or a round-robin load-balancing algorithm. You choose the algorithm associated with each group.

Load balancing is a crucial requirement for many large-scale distributed systems. The OrbixNames approach to load balancing is simple but powerful and highly scalable

Open Interaction with COM

OrbixWeb uses the standard CORBA architecture to provide interoperability between applications running on different platforms and written in different programming languages; for example, a Java application running on Windows can communicate directly with a COBOL application running on OS/390.

OrbixCOMet extends this interoperability to include COM applications. Using OrbixCOMet, COM and CORBA applications can communicate with each other using their native object models. For example, a Visual Basic client running on Windows can communicate with a CORBA server running on UNIX. To the client, the CORBA object appears to be a COM object.

OrbixCOMet is a two-way bridge that translates messages between COM and CORBA applications from one object model to the other. It enables COM applications, written using tools such as Visual Basic, PowerBuilder, Delphi, MS Office or Active Server Pages, to access CORBA applications running on any platform.

OrbixCOMet provides a wide variety of deployment options to suit the requirements of different types of system. For example, you can deploy the OrbixCOMet bridge on each client machine, or on a single, central machine. You can also optimize OrbixCOMet applications for downloading across the Internet.

Deployment Support Tools

Managing a distributed Internet CORBA system requires specialized tools. OrbixWeb includes the following graphical deployment support tools:

- *The OrbixWeb Configuration Explorer*. This tool allows you to configure your installed OrbixWeb components, including the OrbixWeb ORB, OrbixNames, OrbixCOMet, and other IONA products.
- *The OrbixWeb Server Manager*. The server manager allows you to control the availability of servers in your system.
- *The Interface Repository Browser*. This tool allows you to maintain the OrbixWeb Interface Repository, a component of the ORB that provides runtime access to the IDL definitions associated with objects in your system.
- *The OrbixNames Browser*. This browser allows you to control the way clients locate objects in your system.
- *Java Daemon Graphical Console*. This console outputs information on server activation, and on the current running threads. It also enables you to set diagnostics levels dynamically and to run garbage collection.

The OrbixWeb deployment support tools enable you to manage your CORBA systems, through these easy-to-use Java-based graphical interfaces.

Technical Overview of OrbixWeb

This section explains briefly how each of the major components of OrbixWeb works:

- The OrbixWeb ORB
- Orbix Wonderwall
- The Orbix Code Generation Toolkit
- OrbixNames
- OrbixCOMet

The OrbixWeb ORB

The OrbixWeb pure-Java ORB is the communications backbone that connects all applications in a distributed object system. Conceptually, the ORB spans your entire network, including Internet hosts that download client applets.

When an OrbixWeb client calls an operation on a distributed object, the stub code, generated by the IDL compiler, intercepts the call. The stub code constructs a *request*, a structure that contains details of the call, and passes it to the ORB. Included in the request are the operation parameters, which have been converted to the correct format for transmission on-the-wire in a process known as *marshalling*.

As shown in Figure 4 on page 12, the client-side components of the ORB convert the request to an IIOP message. They then transmit this message across the network to the server that hosts the distributed object.

The server-side ORB components pass the message to the object skeleton code, which extracts the message details and calls the required member method on the implementation object. The skeleton code passes any response, such as a return value or a Java exception, to the ORB, which returns the result to the client.

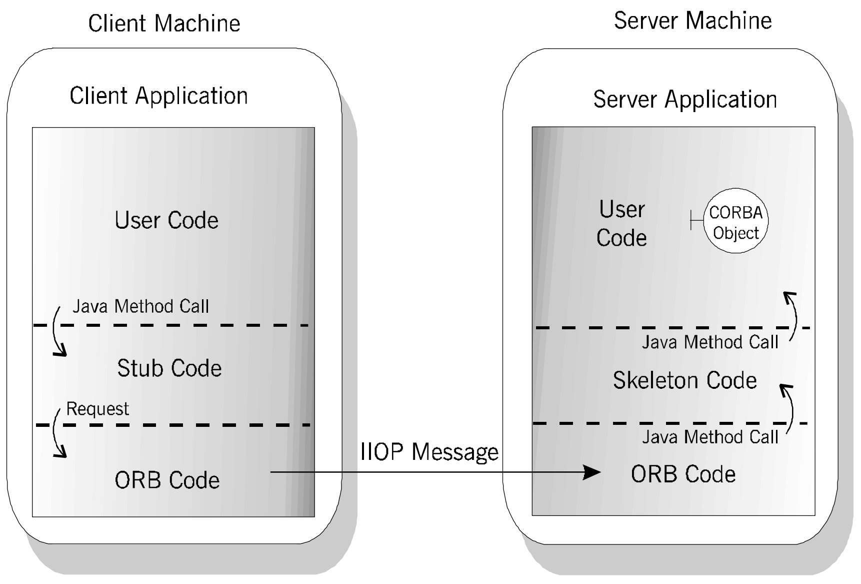


Figure 4. Invoking an Operation on a CORBA Object

Components of the ORB

OrbixWeb implements the ORB using two basic components: the *OrbixWeb runtime* and the *OrbixWeb daemon.* The browser-independent *OrbixWeb runtime* provides much of the ORB functionality, including the full API. This is implemented as a pure-Java class set; and is fully compatible with Netscape, Microsoft and JavaSoft browser technology.

The *OrbixWeb daemon* process runs on each server. The primary role of the OrbixWeb daemon is as a server activator, for both Java and C++ servers. When a client calls an operation on an object, a server process containing the target object must be available. If the process is not running, the OrbixWeb daemon at the server host launches the server automatically.

OrbixWeb also includes a Java version of the server activation component. The *OrbixWeb Java daemon* enables you to launch Java CORBA server components either as new threads within the Virtual Machine running the Java daemon (*in-process*) or alternatively, in separately launched Java VMs (*out-of-process*).

Using the OrbixWeb Java daemon to create server threads within a single VM provides a natural and scalable approach for Java server deployment. In-process activation brings significant scalability and performance benefits—connection time is reduced, and invocations between server objects benefit from co-location. In addition, significantly less memory is required for multiple server configurations because connections are shared.

To manage the server processes running in the system, the daemon uses a database called the *Implementation Repository*. The Implementation Repository maps each server name to the filename of the corresponding executable code.

The OrbixWeb ORB has evolved through many revisions. With each revision, the ORB components have seen improvements in many areas, including performance. The components of the OrbixWeb 3.2 ORB are highly optimized to provide the most rapid and efficient communications with Internet objects.

CORBA Standard Features of the ORB

The CORBA 2.1 specification describes many standard features that a CORBA-compliant ORB must support. These features include:

- *Runtime access to type information*. Normal CORBA programming involves compiling the IDL associated with your objects and use the generated Java code in your applications. This means that your client programs can call only operations on objects whose interfaces are known at compile-time. A more dynamic approach would be to allow clients to locate object type information at runtime.

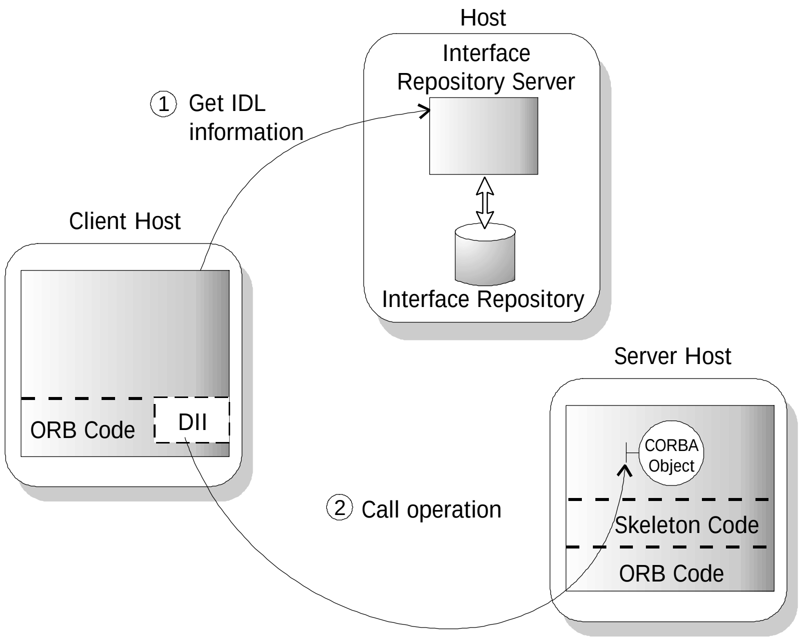


Figure 5. Client Calling an Operation Using the DII

The CORBA *Interface Repository* is a database that stores information about IDL interfaces implemented by objects in your network. At runtime, any OrbixWeb application can query the Interface Repository for this information

- *Dynamic CORBA programming*. A client that obtains information about IDL interfaces at runtime cannot use the normal, static approach of calling operations on those interfaces. This approach is not available because the client has no access to the corresponding stub code. The CORBA Dynamic Invocation Interface (DII) allows clients to call operations on interfaces about which they have no information at compile-time, as shown in Figure 5 on page 13.

  CORBA also supports dynamic server programming. A CORBA server can receive operations through IDL interfaces for which no CORBA object exists. Using an ORB component called the Dynamic Skeleton Interface (DSI); the server can then examine the structure of these operations and implement them at runtime. Figure 6 shows a dynamic client program communicating with a dynamic server program.

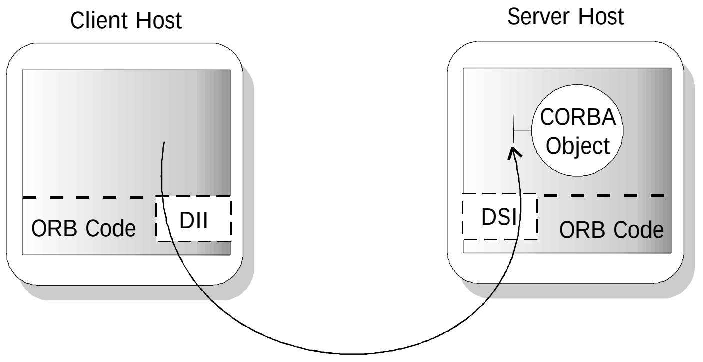


Figure 6. Operation Call Using the DII and DSI

- *Adding application-specific data to IIOP messages*. The IIOP messages that encode operation calls and their results can include embedded, application-specific data. This data is stored in a structure called a *service context*. An ORB makes no attempt to interpret the contents of service contexts, instead it provides you with a mechanism for adding them to and extracting them from IIOP messages.

- *Support for the Internet Inter-ORB Interoperability Protocol*. OrbixWeb 3.2 complies with version 1.1 of the Internet Inter-ORB Interoperability Protocol (IIOP). This allows you to send IIOP messages as fragments. This increases parallelism and improves the overall dispatch speed for very large messages. It also brings the additional benefit of lower memory consumption.

- *Support for multiple ORBs.* OrbixWeb 3.2 provides support for multiple ORBs. This means that each newly created ORB is completely independent from any other ORB. For example, in terms of its configuration, connections, listener ports, server object tables and so on. This gives applications enhanced flexibility, and facilitates complete applet separation in browsers.

OrbixWeb Enhancements to the ORB

In addition to providing the mandatory CORBA 2.1 functionality, OrbixWeb provides extensions that programmers require in building real world applications. These include:

- *A flexible threading model*. Most operating systems provide support for the use of threads. Threads can be viewed as lightweight processes contained within a single operating system process. OrbixWeb provides a fully multi-threaded environment that is critical in building high-throughput application servers.

  OrbixWeb does not tie you to a pre-defined server-side threading model. Instead, it lets you define your own threading model and provides examples of various useful models. These models include creating a thread for each object, creating a thread for each incoming request, and creating a pool of threads to handle all incoming requests.

- *Filters*. OrbixWeb filters allow your applications to intercept messages to and from CORBA objects. This provides a useful mechanism for such tasks as compressing information for application-specific functionality.

- *Loaders*. Objects in the CORBA model are considered constantly available. However, servers can stop and be launched again between operations, potentially losing state information for your objects. OrbixWeb loaders provide a persistence mechanism that allows you to save and load object state information.

- *Smart proxies*. A client calls an operation on a CORBA object using its normal programming language syntax. The object that first receives this call is located in the client address space and is known as a *proxy*. The proxy forwards the corresponding request to the ORB for transmission to the real CORBA object.

  In certain cases, it is useful to create a specialization of the proxy. For example, a specialized proxy could allow caching of data to reduce network traffic. OrbixWeb allows you to create *smart proxies* that extend the behavior of normal proxy objects.

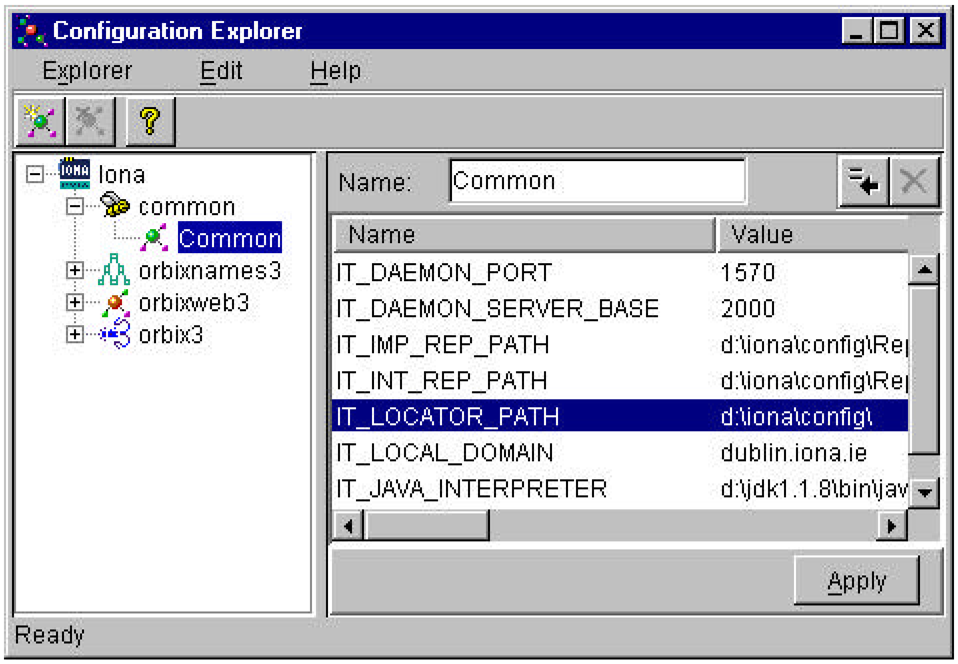
- *Transformers.* OrbixWeb transformers enable you to modify OrbixWeb operation calls immediately before and after transmission across the network.

Using transformers, you can apply custom encryption algorithms to messages sent between OrbixWeb applications.

- *IoCallbacks.* OrbixWeb ioCallbacks allow your applications to be informed when a new connection is established, or when an existing connection is closed. A connection is opened when a client first communicates with the server; and is closed when the client terminates or the communications layer reports a break in service between the server and the client. You can use ioCallbacks to monitor these connections and inform your application when these events occur.

OrbixWeb Configuration

OrbixWeb 3.2 components allow flexibility for configuring for particular deployment scenarios. This is done by exposing key parameters and settings within the OrbixWeb runtime. This allows administrators, for example, to enable or disable parts of runtime functionality, change the JDK version used, alter port numbers, optimize the size of tables used to track server objects, and so on.



Name: Common

Name	Value
IT_DAEMON_PORT	1570
IT_DAEMON_SERVER_BASE	2000
IT_IMP_REP_PATH	d:\iona\config\Re
IT_INT_REP_PATH	d:\iona\config\Re
IT_LOCATOR_PATH	d:\iona\config\
IT_LOCAL_DOMAIN	dublin.iona.ie
IT_JAVA_INTERPRETER	d:\jdk1.1.8\bin\jav

Figure 7. The OrbixWeb Configuration Explorer

You can alter these configuration values programmatically using API calls, by setting environment variables, or by using applet tags. However, the recommended way to configure OrbixWeb is to use the OrbixWeb configuration files. These act as the central repositories for all OrbixWeb configuration information.

The OrbixWeb configuration files can also be configured and included with the downloaded OrbixWeb class-set. This allows remote Internet clients to be configured from the central distribution point.

For ease-of-use, default configuration values ensure minimum configuration and installation overhead. A single graphical Java-based interface ensures that configuration is straightforward. You can modify all aspects of development and deployment using the OrbixWeb Configuration Explorer graphical interface as shown in Figure 7 on page 16.

Orbix Wonderwall

When deploying CORBA applications on the Internet, it is vitally important to prevent insecure access to the distributed objects in your local network. Orbix Wonderwall allows you to establish a fully secure system, based on the Internet firewall model, for application-to-application communication across the Internet and across corporate intranets.

Security for CORBA Objects

A standard approach to Internet security is to use a firewall to restrict access to hosts on your local network. The basic Internet firewall model involves passing all messages to and from the internal network through a single point. The messages are monitored and controlled at this single point.

Using an Internet Firewall

Internet firewalls restrict access to your internal network in a variety of ways; for example, you can use a firewall to restrict access to certain hosts and certain commands. In a distributed object environment such as CORBA, it is important that your firewall supports an object-oriented approach to security. In addition, this security should provide fine-grained control of messages.

Orbix Wonderwall is developed according to the Internet firewall model. It provides a flexible, object-oriented approach to security and allows you to control access to individual objects and individual operations supported by those objects. Orbix Wonderwall is a firewall proxy server that filters, controls, and logs IIOP messages sent between CORBA applications on the Internet and applications on your internal network.

Using a Internet Firewall on a Bastion Host

Usually, an Internet firewall restricts access to a single IP port on a single host for each service supported externally; for example, HTTP access or SMTP access. A refinement of this model uses a dedicated, secure host for the firewall. This type of dedicated host is known as a *bastion host*.

The bastion host is dedicated exclusively to the role of gateway and its configuration can be optimized to provide the highest level of security. A clear advantage of this approach is that security effort can be focused on a single machine.

An Internet firewall is usually implemented as a proxy server process that runs on the bastion host. For example, this is the case with HTTP and SMTP firewalls. The role of the server process is to listen to incoming messages on a well-known IP port and to pass on these messages to the internal network, after subjecting them to close scrutiny. When forbidden or potentially hostile messages are encountered, these are blocked and not passed on. Orbix Wonderwall also runs as a proxy server process in this way, examining incoming and outgoing IIOP messages.

Using Orbix Wonderwall and HTTP

The applet security built into the Java Virtual Machine in all browsers restricts connections to the host from which the applet was downloaded. While this basic principle is sound, the current implementation is over-restrictive and can cause problems for developers. By delivering your applets from the Wonderwall, using its optional HTTP server functionality, you can permit your applets to contact all the relevant objects, regardless of which host they reside on, while still retaining Orbix Wonderwall's security, filtering and logging functionality.

If you wish to use the Wonderwall to handle your intranet access to CORBA objects, Orbix Wonderwall provides a zero-configuration set-up, purely to work around this applet security restriction. Orbix Wonderwall can run with no intervention or configuration whatsoever.

In addition, Orbix Wonderwall s full HTTP 1.1 implementation enables applets access to CORBA server objects, even from behind a client s corporate firewall. *HTTP tunneling* allows IIOP requests destined for the Wonderwall to be wrapped up in HTTP by the applet s ORB classes.

How Orbix Wonderwall Works

IIOP specifies the way in which CORBA messages are encoded for network transmission. IIOP makes it possible for clients of one ORB to call operations on objects created with another ORB and to interpret correctly any returned results.

To access a remote object, a client requires a reference to the object. In IIOP, each object has an associated Interoperable Object Reference (IOR), which stores addressing information for the object, including:

- The host on which the object runs.
- The port on which the object listens for messages.
- An object key, which is a series of bytes that identify the object.

When a client uses an IOR, the client-side ORB opens a TCP/IP connection to the object host and port number. The client and object transfer IIOP messages across this connection. If multiple objects use the same host and port, the client can use the same connection to communicate with the other objects.

In the IIOP model, two main types of message pass between clients and objects: *Requests* and *Replies*. An IIOP Request is a message that encodes an operation call from a client to an object. An IIOP Reply is a message sent from an object to a client and contains the results of an operation call.

Orbix Wonderwall filters Request messages based on the information stored in a particular part of the message structure. This part of the message is the *header*, and it includes the following information:

- The object key of the target object.
- The name of the operation being called.
- The IP address of the client.
- The user name under which the client is running.

The Orbix Wonderwall server runs on the bastion host and listens for Request messages on a specified TCP/IP port. You control how the server filters these messages by modifying a simple configuration file. Using this file, you can specify a flexible set of rules for allowing or denying access to certain objects or operations. If a given Request message meets your criteria for access to the internal network, the Orbix Wonderwall server forwards it to the appropriate CORBA object.

Features of Orbix Wonderwall

There are potential security risks when complex applications connect to the Internet. Using Orbix Wonderwall protects your internal network from external security risks. The Orbix Wonderwall server is a simple standalone process, which interacts with the bastion host in a simple way.

The Orbix Wonderwall server has the following features that contribute to the security of your network:

- *Supports use of a bastion host*. The Orbix Wonderwall security model supports the use of a bastion host as the foundation of your Internet firewall. You can install the Orbix Wonderwall server on a bastion host or a regular host.
- *Filters messages based on Request header*. A number of message types are defined for the IIOP protocol and any or all of these can be blocked if necessary. Orbix Wonderwall provides comprehensive filtering of Request messages based on the content of the message header. Request messages are checked rapidly and passed to the internal network with little performance overhead.
- *Supports message encryption*. Message encryption is an essential feature needed for the exchange of private messages over the Internet. Orbix Wonderwall now includes embedded support for IIOP messages encrypted using Secure Sockets Layer (SSL) encryption. The IONA product OrbixSSL allows you to add this encryption to your OrbixWeb applications simply by making some basic configuration changes.

  Orbix Wonderwall also allows you to encrypt IIOP messages using an OrbixWeb feature called *transformers*. Using transformers, you can apply a custom encryption algorithm to messages sent between OrbixWeb applications.
- *Provides fine-grained control of security*. Orbix Wonderwall provides the kind of fine-grained control of security needed for a distributed object environment. For example, it allows you to control access to individual objects, and to permit or deny access to specific methods defined on that object.
- *Supports message logging*. The logging facility of Orbix Wonderwall is a powerful facility for tracing the history of suspicious message exchanges. It is also generally useful as a debugging and monitoring facility.
- *Blocks messages unless specifically allowed*. Orbix Wonderwall observes and promotes good security practice. For example, all messages are forbidden unless you expressly allow them.

The Orbix Code Generation Toolkit

The Orbix Code Generation Toolkit is a rapid application development tool for CORBA programmers. The toolkit allows you to create scripts that generate code from an IDL file. For example, you can generate Java or C++ code based on your IDL, or convert your IDL files to formats such as HTML, RTF, or LaTeX. The toolkit also includes a set of ready-to-use scripts that address many repetitive CORBA programming tasks.

How the Code Generation Toolkit Works

As shown in Figure 8, an IDL compiler typically contains three major components. A *parser* processes an input IDL file and constructs an in-memory representation called a *parse tree*. A *back-end processor* then traverses this parse tree and generates, for example, JAVA stub code.

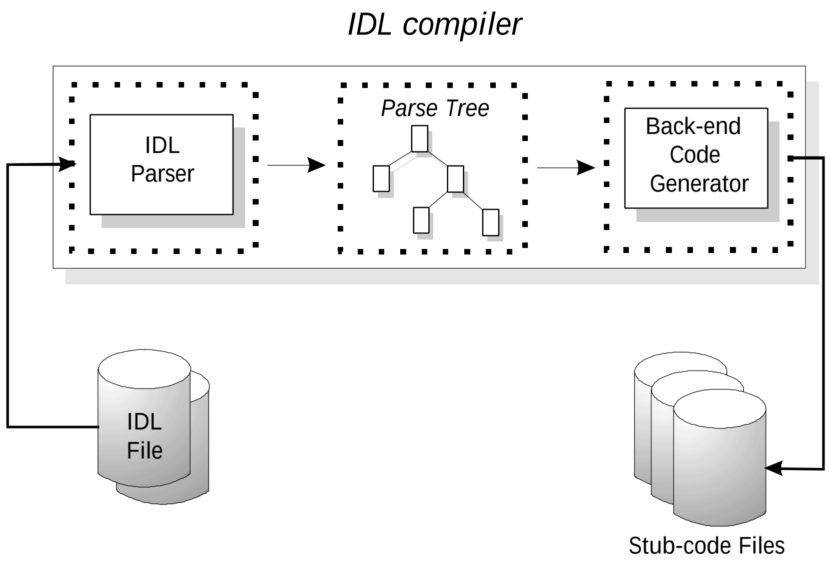


Figure 8. Structure of an IDL Compiler

The core component of the Orbix Code Generation Toolkit is an executable called IDLgen, which employs a variation on this architecture. IDLgen uses an IDL parser and creates a parse tree, but instead of having a back-end that generates stub code, it has a back-end interpreter for the scripting language Tcl.

The Tcl interpreter inside IDLgen has been extended so that you can manipulate the IDL parser and parse tree with Tcl commands. This makes it possible to customize the output from IDLgen using a Tcl script, as shown in Figure 9.

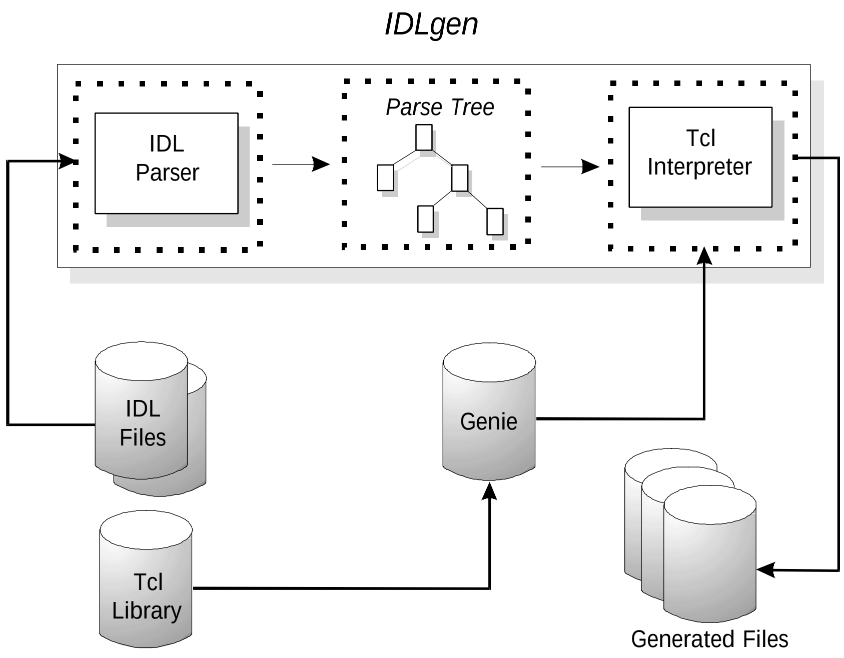


Figure 9. Using a Tcl Script to Customize IDLgen Output

Orbix code generation scripts, also known as *genies*, are Tcl scripts that use IONA's interpreter extensions to create code based on IDL files. Typically, these scripts instruct IDLgen to parse an IDL file, manipulate the resulting parse tree, and then generate code.

The Orbix Code Generation Toolkit is supplied with a set of ready-to-use code generation scripts. Simply by running these scripts, you can perform common tasks such as creating a complete client-server application for an IDL file, creating commands that print your IDL data types, and so on. The toolkit also includes libraries of Tcl procedures that allow you to create your own customized code generation scripts.

Using the Code Generation Tools

The ready-to-use code generation scripts allow you to speed up your development process without any knowledge of Tcl. Each script takes an IDL file as input and generates code based on the content of the file. The toolkit includes scripts that generate:

- *A complete client-server application*. By generating a complete demonstration application from your IDL definitions, the toolkit allows you to kick-start a new project in just a few minutes. The more complex your IDL definitions, the more time you save.

- *Code for new operations in an IDL interface*. When your definitions of IDL operations change, so must the Java code that implements them. The toolkit includes a script that allows you to generate the Java method signatures that correspond to IDL operations.

- *Commands that assign random values to IDL types*. These commands allow you to develop a prototype application rapidly. For example, consider a server that queries a database and returns data to a client. During initial development work, you can replace the server data retrieval with these random value commands.

- *Commands that display the values of IDL types*. These commands can print simple data type values and recursively examine compound data types, such as structs, unions, and sequences. These print commands can be an excellent debugging aid throughout the lifecycle of a project.

- *Commands that test IDL types for equality*. Testing IDL types for equality is a common requirement that is not always easy to implement. These functions can test the equality of values of any type in your IDL file.

These scripts dramatically reduce development time by automating many repetitive coding tasks. They also improve programmer skills by illustrating common programming tasks for CORBA applications.

In summary, the ready-to-use code generation scripts allow you to get a project started quickly, to learn CORBA programming by example, and to debug and maintain applications. However, the uses of code generation scripts are not confined to application development. The toolkit also includes example scripts that do the following:

- *Generate a statistical analysis of an IDL file's contents*. This script displays a number of statistics about your IDL definitions, such as the number of times each IDL construct is used.

- *Convert IDL files into HTML files*. This script produces an HTML version of your IDL, with hypertext links for easy navigation.

Expanding the Toolkit

The ready-to-use scripts supplied with the Orbix Code Generation Toolkit provide a powerful, general-purpose way to improve your productivity. By creating your own scripts, you can extend this approach to other coding tasks that are specific to your development effort.

The Tcl language is implemented as a library of C functions. A hash table maps the name of a Tcl command to the C function that implements that command. This scheme allows programmers to write C functions for new commands and register them with the Tcl interpreter. IONA provides a library of such extensions that makes creating new code generation scripts a straightforward task.

To expand the Orbix Code Generation Toolkit, you write new Tcl scripts that use IONA's extensions to the Tcl interpreter. These extensions include:

- *IDL parsing*
- *Mapping IDL to programming languages*
- *General extensions*

The rest of this section describes provides an overview of these extensions. Of course, when learning to use the toolkit, you do not have to write your code generation scripts from scratch. You can start by taking the ready-to-use scripts and modifying them.

IDL Parsing

The built-in IDLgen command `idlgen_parse_idl_file` parses an IDL file. It takes two parameters: the first is the name of the IDL file, and the second is a list of directives that are passed to the IDL preprocessor.

Once an IDL file has been preprocessed by the parse command, the root of the parse tree is placed into a global array variable, `$idlgen(root)`. The parse tree is a representation of the IDL, with each node in the tree representing an IDL construct. For example, consider the following IDL file:

```
interface Account {
    readonly attribute long accountNumber;
    readonly attribute float balance;
    void makeDeposit(in float amount);
};
```

```
interface Bank {
    Account newAccount();
};
```

The parse tree for these definitions is shown in Figure 10.

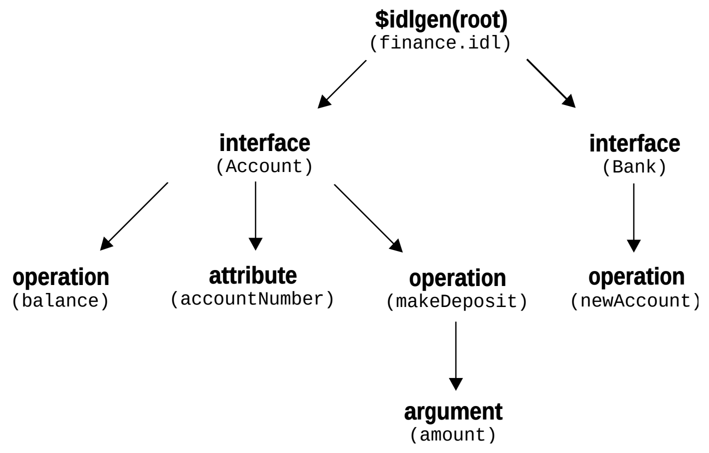


Figure 10. Example of an IDL Parse Tree

Mapping IDL to Programming Languages

When writing a script, you can traverse the IDL parse tree and examine information associated with each node. For example, a node associated with an operation can provide such information as the operation name, its return type, and so on. You can then use this information when generating the output from the script.

The Tcl commands that traverse and extract information from the parse tree provide the core functionality of IDLgen. Using these commands, you can completely or selectively traverse the parse tree, repeating operations for particular types of IDL construct. For example, each time you encounter an IDL operation, you might want to generate a Java method that implements that operation.

The Code Generation Toolkit simplifies many of the common tasks involved in processing IDL files in this way. For example, it allows you to configure how your generated code is indented, it maps names of IDL constructs to Java or C++ according to standard CORBA rules, it generates the correct memory management code for particular constructs, and so on.

General Extensions

The Code Generation Toolkit contains many general-purpose extensions to Tcl. These provide clear improvements to Tcl that speed up script development. For example, the Tcl `source` command allows you to use commands defined in another Tcl script, but cannot use a search path to locate the script. The toolkit defines an equivalent command, called `smart_source` that can use a search path to locate a script. Other improvements include a flexible mechanism for writing blocks of text to output files.

OrbixNames

OrbixNames is IONA's implementation of the CORBA Naming Service. This service is used in distributed applications to allow clients to locate the objects they require.

Location Transparency with OrbixNames

The CORBA Naming Service acts as a central repository of objects that clients can use to locate server applications. A server that holds a CORBA object can instruct the Naming Service to associate a particular name with that object. A client then locates the object by asking the Naming Service to look up the corresponding name.

The ability to locate objects is fundamental to clients in a distributed system. Before using an object, a client must get a reference to it. A brute force approach to locating objects would allow the client to specify the host on which the object's server runs, the port the server listens on, an identifier for the object within that server, and so on. But such an approach has an important drawback. If the location of the object changes, so too must the client code.

The Naming Service provides *location transparency* for objects, because the names associated with them are abstract. An object name is totally independent of properties of the object, such as its location in the network. CORBA specifies the structure of object names, but you can decide their content.

The Format of Object Names

In the Naming Service, object names are structured as a graph. The entry point to this graph is a single root node that provides access to all the names in the Naming Service.

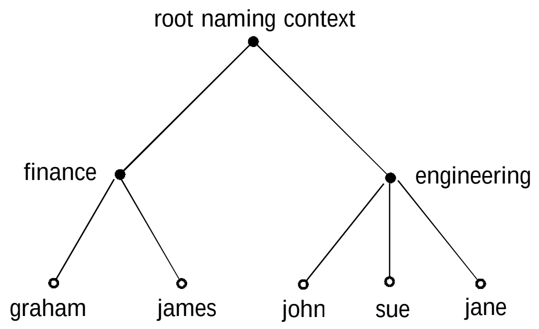


Figure 11. An Example Naming Graph

Figure 11 shows an example of simple naming graph. The graph contains two types of node:

- *Naming contexts*, represented as full circles in Figure 11, act as containers for other nodes. They act in a similar way to directories in a file system. The node at the root of the graph is called the *root naming context*.
- *Names*, represented as circle outlines in Figure 11, are associated directly with application objects. In this way, they are analogous to filenames in a file system.

A name that contains one or more naming contexts is known as a *compound name*. The full compound name of an object represents a full path from the root naming context to the object’s name node. Continuing the file system analogy, this is similar to a fully qualified filename.

How OrbixNames Implements the Naming Service

OrbixNames implements the Naming Service as a normal CORBA server. Clients and servers communicate with OrbixNames through a standard set of IDL interfaces, defined by the OMG. The OrbixNames server stores names and references to the associated objects in a database, as shown in Figure 12 on page 28.

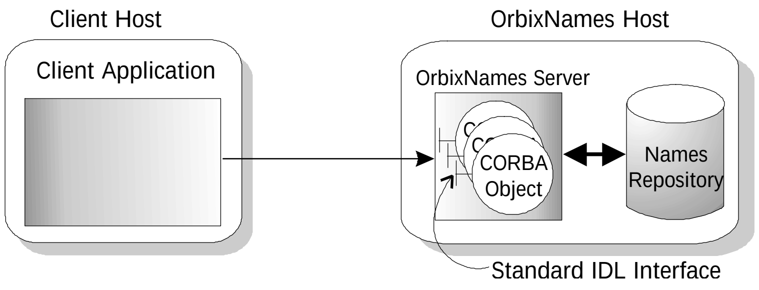


Figure 12. Using the OrbixNames Server

The standard IDL interfaces to the OrbixNames server are straightforward. For example, they include operations to:

- Create a new naming context.
- Bind a name to an object.
- Look up an object name. Remove the association between a name and an object.
- Remove a naming context.

When deploying OrbixNames, you can choose between using a single OrbixNames server to manage the database of names or you can use a federation of databases, each managed by a single server. Whether you choose to deploy a single OrbixNames server or a federation of servers, OrbixNames works efficiently at all system scales.

Load Balancing with OrbixNames

The Naming Service provides an important opportunity to balance the load of client requests across several objects. For example, consider an application in which two objects, called A and B, can provide exactly the same service to clients. To balance the load, some clients could look up the name A, and use that object, and others could look up the name B.

The problem with this system, however, is that the load balancing logic is hard coded in the client applications. If the load balancing requirements change, the client code must also change. In a large-scale application, where clients are deployed very widely, such a system would be extremely difficult to administer.

OrbixNames overcomes this scalability problem by extending the functionality of the Naming Service. In the standard CORBA Naming Service, a name maps to one object only. In OrbixNames, a name can also map to a group of objects. An *object group* is a collection of objects that can increase or decrease in size dynamically. For example, {A, B} would constitute an object group.

In OrbixNames, each object group has a selection algorithm. When a client resolves a name associated with an object group, OrbixNames picks an object from the group using the appropriate algorithm. Two algorithms are supported: random selection and round-robin selection. These algorithms are simple, but in practice they are very effective.

Using Object Groups

OrbixNames supports object groups by introducing new IDL interfaces to the OrbixNames server. These interfaces enable you to create object groups, add objects to and remove objects from groups, and to find out which objects are members of a particular group. If you want to take advantage of object groups, you can use these interfaces in your servers to create and manipulate groups. Your client code remains unchanged.

Figure 13 illustrates the concept of associating a name with multiple objects using an object group.

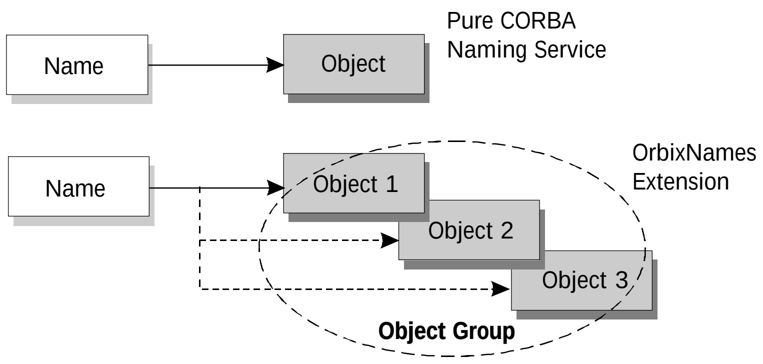


Figure 13. Associating a Name with an Object Group

A client looks up the name of an object group using the normal Naming Service interface. The client is not aware that the name is associated with a group, rather than a single object. Using the CORBA standard interface to the Naming Service is very important to clients, because they are generally much more difficult to update than servers.

OrbixCOMet

OrbixCOMet allows COM applications to communicate directly with CORBA applications. With OrbixCOMet, programmers can use familiar Windows-based languages to write applications that communicate with powerful UNIX, OS/390, and Windows servers through CORBA.

COM and CORBA Working Together

OrbixCOMet is IONA's implementation of the OMG COM/CORBA Interworking specification. This specification defines a model for transparent, two-way interworking between CORBA and COM/Automation.

Two-way interworking means that CORBA and COM/Automation applications integrate seamlessly. For example:

- A COM or Automation client can call objects in a CORBA server. Because both COM and CORBA support distribution, the client and the server can be on different machines.
- A CORBA client can call objects in a COM or Automation server. Again, the client and the server can be on different machines.

Transparency in the interworking mechanism means transparency between the COM and CORBA object models. For example:

- A client working in the CORBA model can view and treat a COM or Automation object as if it were a CORBA object. This is because the object has an OMG IDL interface that the CORBA client can understand.
- A client working in the COM model can view and treat a CORBA object as if it were a COM or Automation object. This is because the CORBA object has a MIDL interface that the COM or Automation client can understand.

Transparency allows clients to work with their familiar object model. They do not have to know that the objects they are using belong to another object system.

The Interworking Model

The COM/CORBA Interworking specification defines the interworking model that specifies how the integration between the COM and CORBA object models is achieved.

In the interworking model, a client in one object system wishes to send a request to an object in the other system. A component called a *bridge* acts as the intermediary between the two systems. This bridge maps messages from one system to the other. It does this transparently so that the client can make requests in its familiar object model.

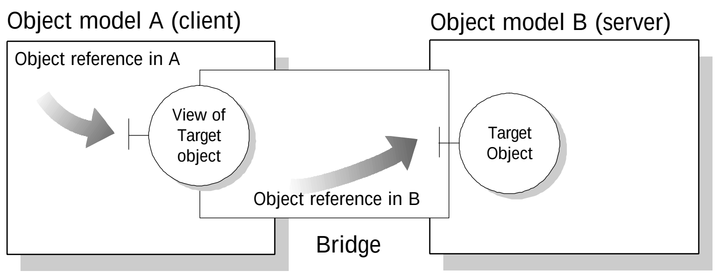


Figure 14. The COM/CORBA Interworking Model

To implement the bridge, the model provides an object called a *view* in the client's system as shown in Figure 14. The view object exposes the interface of the target object in the model understood by the client.

The client makes requests on the view object. The bridge maps these requests into requests in the server's object model and forwards these requests to the target objects across the system boundary. The workings of the bridge are transparent to the client.

Using OrbixCOMet

Figure 15 shows the components involved in the OrbixCOMet implementation of the interworking model.

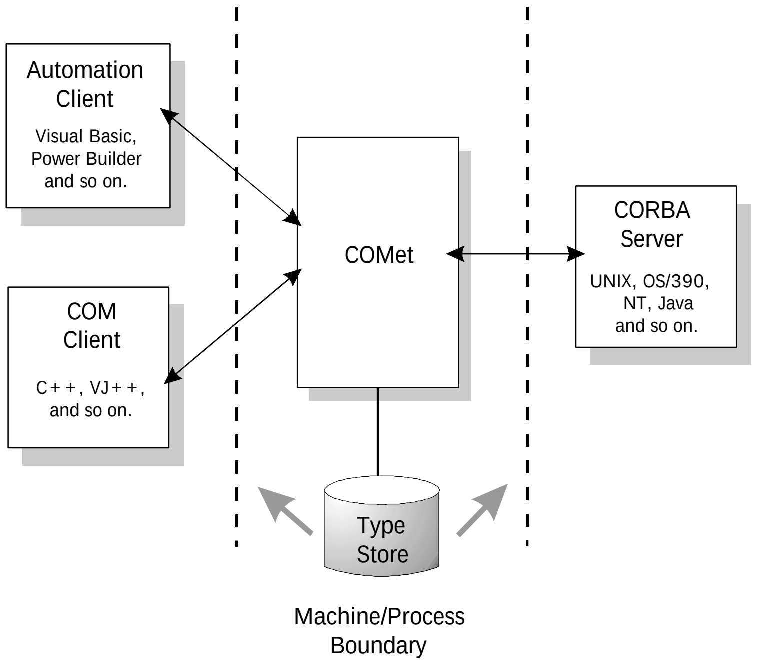


Figure 15. How OrbixCOMet Implements the COM/CORBA Interworking Model

In this figure, COM and Automation clients call operations on a CORBA object, but of course, CORBA clients can also access COM objects . The OrbixCOMet bridge is implemented as a set of DLLs that are capable of dynamically mapping between the two object models. The bridge is not involved in requests sent between clients and servers of a single object model.

The OrbixCOMet bridge uses another OrbixCOMet component, called *the type store*. This component provides information to the bridge about all the COM and CORBA types in your system. COM types are defined using MIDL. CORBA types are defined using OMG IDL. The OrbixCOMet type store holds a cache of your MIDL and IDL type information, stored in a neutral binary format.

At runtime, OrbixCOMet uses your COM type libraries and the CORBA Interface Repository to obtain information about the types defined in your system. To improve performance, the type store includes a sophisticated caching system that prevents unnecessary checking of type libraries or the Interface Repository.

The OrbixCOMet *type store manager* is a graphical tool that you can use to control and manage the contents of the type store. OrbixCOMet also includes an equivalent set of command-line utilities.

Deploying OrbixCOMet Applications

OrbixCOMet supports communication using either the DCOM protocol or CORBA IIOP. Therefore, when it comes to deploying your applications, you have a great degree of flexibility in how you choose to install OrbixCOMet.

There are two basic deployment models for OrbixCOMet applications:

- Install the bridge on each client machine. In this case, all client-server communications use the protocol associated with the server application. For example, if the server is a CORBA server, all communications use IIOP.
- Install the bridge on a central server. In this case, the client communicates with the bridge using the client's associated protocol and the bridge communicates with the server using the server's associated protocol.

To deploy an OrbixCOMet application on the Internet, you embed an Active X control in an HTML page. When doing this, you can choose from the following options:

- Download the entire OrbixCOMet bridge to the client machine. To do this, you can bundle the bridge files, for example, in a single CAB file. In this case, your Active X control communicates with your Internet server using IIOP.
- Download a single, small DLL and leave the bridge on the Internet server. In this case, your Active X control communicates with your Internet server using DCOM.

These flexible deployment models combine with the other advantages of OrbixCOMet to make an OrbixCOMet system very easy to manage. The dynamic nature of the OrbixCOMet bridge is a very important factor in this. The bridge maps types at runtime, without the use of static stub code. Consequently, if you want to update existing servers or replace a DCOM server with a CORBA server, your existing clients can continue to work without any modification.

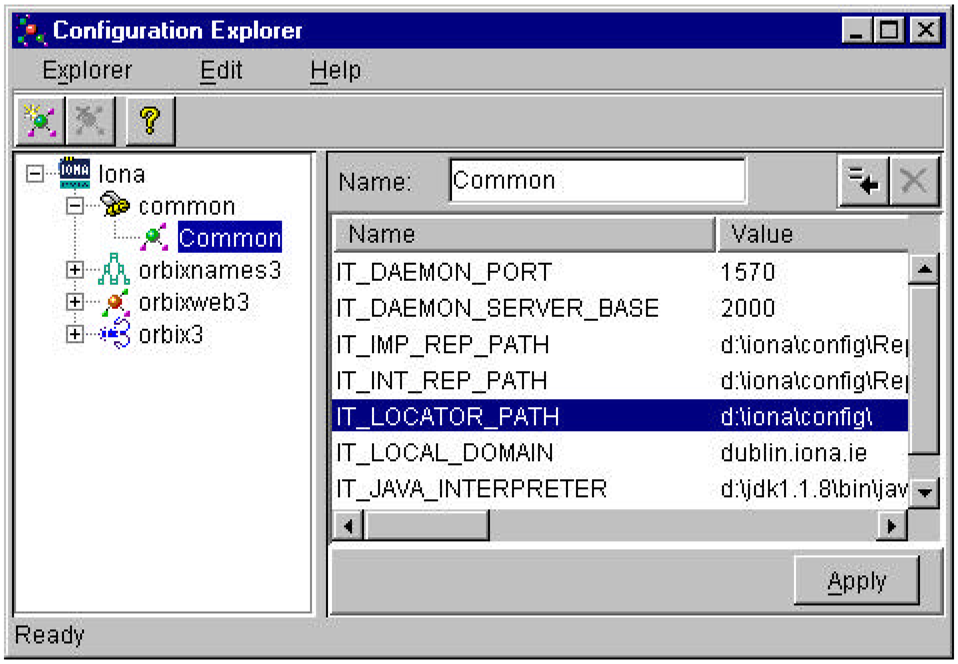
New Features in OrbixWeb 3.2

OrbixWeb 3.2 has many new features that enhance its proven ease of use, flexibility, reliability, and scalability. This section describes the major areas of improvement in this release.

New Configuration Mechanism

All components of OrbixWeb now share a single, unified configuration mechanism. The configuration variables for each component are defined in a separate *configuration scope*, making it easy to understand where the variables apply. A new configuration file format makes it easy to update multiple variables simultaneously.

The new OrbixWeb configuration explorer supports this configuration mechanism. As shown in Figure 16, this tool provides a simple graphical interface that allows you to manage the configuration values in each OrbixWeb configuration scope.



Name	Value
IT_DAEMON_PORT	1570
IT_DAEMON_SERVER_BASE	2000
IT_IMP_REP_PATH	d:\iona\config\Re
IT_INT_REP_PATH	d:\iona\config\Re
IT_LOCATOR_PATH	d:\iona\config\
IT_LOCAL_DOMAIN	dublin.iona.ie
IT_JAVA_INTERPRETER	d:\jdk1.1.8\bin\jav

Figure 16. Managing OrbixWeb Configuration Values

Future releases of other IONA products will support the new configuration mechanism, and can be managed using the explorer. If you use the OrbixWeb configuration syntax for your own applications, you can also use the explorer with these.

A new command-line utility, called `dumpconfig`, provides a complete description of your current configuration. This utility outputs all the configuration values used by your OrbixWeb application, and the location of the OrbixWeb configuration files in your system. It also reports any syntax errors in your configuration files that might go unreported by OrbixWeb. This information is extremely useful when debugging applications, to ensure that your configuration settings are what you expect.

New Code Generation Tools

The Orbix Code Generation Toolkit is an entirely new feature in OrbixWeb 3.2. The toolkit automates many common tasks for OrbixWeb developers. Version 3.0.1 introduces support for Java code generation. Refer to "The Orbix Code Generation Toolkit" on page 21 for details.

New Examples

Every OrbixWeb installation includes many demonstration programs, each illustrating a particular feature of OrbixWeb or a common coding task. In OrbixWeb 3.2, existing examples have been extensively rewritten to improve clarity and the potential for reuse, and several new examples have been added for each OrbixWeb component.

JDK 1.2 Support

The OrbixWeb 3.2 runtime now includes support for Sun's Java Development kit 1.2 (Java 2 Platform).

Improved SSL Support

OrbixWeb 3.2 provides support for Secure Sockets Layer (SSL) message encryption. SSL support has now also been added to the OrbixNames, Orbix Wonderwall and OrbixCOMet components of the OrbixWeb 3.2 Professional Edition. The OrbixWeb 3.2 graphical tools have also been SSL-enabled.

You must have OrbixSSL for Java installed in order to use SSL message encryption with OrbixWeb.

Improved IDL Support

OrbixWeb 3.2 now fully supports the IDL data type `fixed`. This was not supported in previous versions of OrbixWeb.

Improved Naming Service Support

The OrbixNames 3.0.1 server, which implements the CORBA Naming Service, is now a multi-threaded Java application. This increases the scalability, performance and deployment portability of this service.

A new feature of OrbixNames enables you to maintain the contents of your Naming Service, even when your applications fail to work correctly. In some cases, a failed application can create naming contexts that have no associated name. Such contexts have associated entries in the Naming Service that are very difficult to remove. OrbixNames places these entries in a special naming context, called the `lost+found` context that allows you to remove them easily.

Interaction with COM

Previous versions of IONA products integrated COM and CORBA using a static ActiveX bridge. OrbixCOMet is a new and superior approach to COM/CORBA interworking. OrbixCOMet is a fast, dynamic, bi-directional bridge that implements the OMG COM/CORBA Interworking specification.

Version 3.0.1 of OrbixCOMet has many enhancements that further improve interaction between COM and CORBA. For example, this release includes:

- Support for SSL message encryption.
- Seemless support for directly binding to COM servers.
- Lifecycle support for CORBA views of COM objects.
- Support for file monikers.
- Improved type store management.
- Enhanced IDL generation from the type store.
- Read-only mode for the type store.
- A single Interface Repository binary, shared by the ORB and OrbixCOMet.
- Enhanced debug and trace information.

New APIs for Application Initialization

OrbixWeb 3.2 introduces new APIs that perform initialization functionality that is specific to your application. These APIs are termed *ORB initialisors*. An ORB initialisor is a Java class that is registered with the ORB and performs application-specific-initialization when the ORB is initialized.

TowerJ Support

TowerJ is a product from Tower Technologies that converts your Java class files into native executables. This enables them to run as fast as native code. OrbixWeb 3.2 provides a Java class that loads a native library, giving the OrbixWeb runtime access to environment variables needed to run native executables. Converting Java class files to native executables can increase the performance of Java servers significantly.

For detailed information on the benefits of using TowerJ, refer to the Tower Technologies web site: `http://www.towerj.com/`

Glossary

ActiveX. A term used by Microsoft to describe a broad set of COM-based technologies. An ActiveX control is a component created in an ActiveX environment that can be downloaded across the Internet.

bastion host. A host dedicated to providing firewall security between an external and an internal network.

client. In CORBA, a program that requests services from a CORBA object.

COM. Component Object Model. Microsoft's framework for developing and supporting objects that can be accessed by any compliant application.

CORBA. Common Object Request Broker Architecture. A standard architecture, defined by the OMG, for communications between distributed objects. An ORB is the core element of the wider OMG framework for developing and deploying distributed components.

daemon. A program that runs continuously and handles periodic service requests.

DCOM. Distributed Component Object Model. An extension of COM that allows COM objects to be distributed across a network.

DES. Data Encryption Standard. A private-key-only (also known as symmetric) encryption scheme that uses a 56-bit key to encrypt and decrypt messages.

DII. Dynamic Invocation Interface. The component of an ORB that allows clients to access CORBA objects without knowing the object interface at compile-time.

DSI. Dynamic Skeleton Interface. The component of an ORB that allows a server to receive and process requests for IDL interfaces, without knowing those interfaces at compile-time.

extranet. An intranet that is extended to strategic users outside a company, such as partners or suppliers.

firewall. A system that protects the resources of a private network from unauthorized access by users of other networks.

genie. In the Orbix Code Generation Toolkit, a genie is a Tcl script that generates text, for example, Java code, based on the contents of an IDL file.

IDL. Interface Definition Language. The CORBA standard language that allows a programmer to define the interfaces to CORBA objects. Clients use these interfaces to access objects across a network.

IIOP. Internet Inter-ORB Protocol. A CORBA standard protocol for communications between distributed applications. IIOP is defined as a protocol layer above TCP/IP.

Implementation Repository. The component of an ORB that stores definitions of servers available in a distributed system.

Interface Repository. The component of an ORB that stores IDL definitions for objects in a distributed system. A client can query this repository at runtime to determine information about an object's interface, and then use the DII to make calls to the object.

Internet. A global, decentralized computer network. Unlike an intranet, access to the Internet is unlimited.

intranet. A network, based on TCP/IP protocols, used to share information in an organization. An intranet is usually accessible only by the organization's members or employees.

legacy application. An existing application that uses languages, platforms, or techniques earlier than the current technology.

marshalling. The process of converting native programming language data types to a format suitable for transmission across a network.

middleware. Any software that connects two or more otherwise separate pieces of software.

multithreading. A thread is part of a process that can run concurrently with other parts. Multithreading is a programming technique in which multiple threads are created in a single program.

object. In object-oriented programming, a single software entity that consists of both data and procedures that manipulate that data. In CORBA, objects can be located anywhere in a network. The functionality of a CORBA object is accessed through interfaces defined in IDL.

OMG. Object Management Group. A consortium that aims to define a standard framework for distributed, object-oriented programming. The OMG is responsible for the CORBA specification.

operation. The IDL equivalent of a function or method. Operations are defined in IDL interfaces and can be called on CORBA objects.

ORB. Object Request Broker. An ORB is a middleware component that acts as an intermediary between a client and a distributed object. The ORB is responsible for delivering messages between the client and object across a network. The ORB hides the underlying complexity of the distributed system, such as differences in hardware, operating systems, and programming languages, from the application programmer.

RC4. This is an encryption scheme based on the use of random permutation and it can be used to provide encryption for SSL secure connections. It was developed by Rivest for RSA Data Security, Inc.

server. A program that provides services to clients. CORBA servers act as containers for CORBA objects, allowing clients to access those objects using IDL interfaces.

SSL. Secure Sockets Layer. A protocol designed by Netscape Communications Corporation to provide encrypted communications on the Internet. SSL is layered beneath application protocols such as HTTP, SMTP, Telnet, and FTP, and is layered above the TCP/IP connection protocol.

Tcl. Tool Command Language. A powerful interpreted programming language that can be easily extended using libraries of custom Tcl procedures. Tcl is used in the Orbix Code Generation Toolkit.

TCP/IP. Transmission Control Protocol/Internet Protocol. TCP/IP is the basic suite of protocols used to connect hosts to the Internet, intranets, and extranets.

unmarshalling. The conversion of data, received over a network, from its on-the-wire representation to data types appropriate to the receiving application.

wrapping. The technique by which a standard interface is applied to a legacy application, thus integrating the application into a wider system.

X.509 Certificate. Used in connection with asymmetric cryptography (public-key cryptography), the X.509 certificate is a standard format for circulating public keys which enables the owner of the public key to be identified.

Further Reading

1. IONA Technologies. *OrbixWeb 3.2 Release Notes*, July 1999[1].

2. Object Management Group (OMG). *The Common Object Request Broker: Architecture and Specification, Revision 2.1*. OMG document number 97-09-01. August 1997[2].

3. Object Management Group (OMG). *CORBAservices: Common Object Services Specification.* OMG document number 98-12-09. March 1995.

4. Baker, Seán. *CORBA Distributed Objects: Using Orbix*. Addison-Wesley, November 1997.

5. Geraghty, Ronan, and others. *COM/CORBA Interoperability*. Prentice Hall, January 1999.

6. Slama, Dirk, and others. *Enterprise CORBA*. Prentice Hall, March 1999.

[1] OrbixWeb release notes are available from the following location:
`http://www.iona.com/online/support/update/index.html`

[2] OMG documents are available from the following location:
`http://www.omg.org`

IONA Technologies

Making Software Work Together™

Contact Details

IONA Technologies PLC
The IONA Building
Shelboune Road
Dublin 4
Ireland

Phone:	+353 1 637 2000
Fax:	+353 1 637 2888

IONA Technologies Inc.
200 West St
Waltham, MA 02451
USA

Phone:	+1 781-902-8000
Fax:	+1 781-902-8001

IONA Technologies Japan Ltd.
Aoyama KK Bldg 7/F
2-26-35 Minami Aoyama
Minato-ku, Tokyo
Japan 107-0062

Phone:	+813 5771 2161
Fax:	+813 5771 2162

Support:	support@iona.com
Training:	training@iona.com
Orbix Sales:	sales@iona.com
IONA s̀ FTP site	ftp.iona.com

World Wide Web: http://www.iona.com/