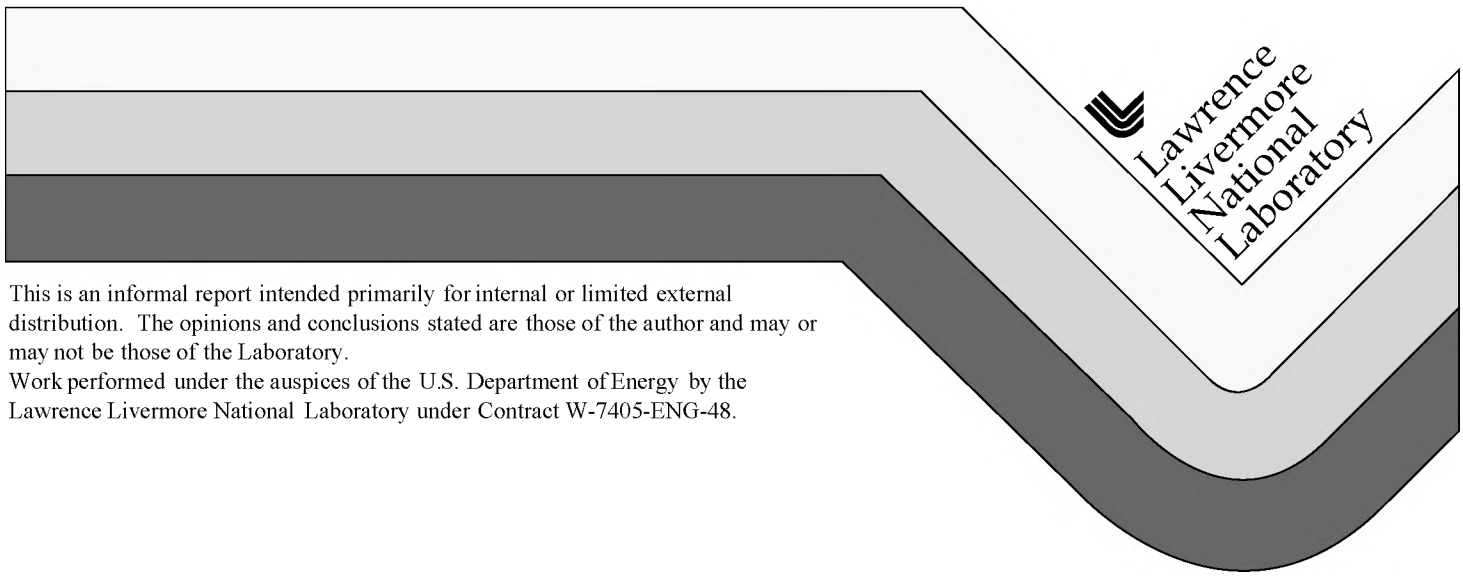
UCRL-ID-133254

Evaluation of CORBA for Use in Distributed Control Systems

F. W. Holloway

February 18, 1999



Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under Contract W-7405-ENG-48.

Evaluation of CORBA for Use in Distributed Control Systems

FY98 LDRD Project

F. W. Holloway

February 18, 1999

Table of Contents

1. Introduction

The Common Object Request Broker Architecture (CORBA)-based Simulator was a Laboratory Directed Research and Development (LDRD) project that applied simulation techniques to explore critical questions about advanced distributed control system architectures.

A three-prong approach comprised of a study of object-oriented distribution tools, computer network modeling, and simulation of key control system scenarios was used in the LDRD project. This input report describes the first of the three approaches — the study of object-oriented distribution tools together with measurements, and predictions of use within the National Ignition Facility (NIF) and some aspects of CORBA which remain to be resolved.

For the last several years LLNL has been developing the Integrated Computer Control System (ICCS), which is an abstract software framework for constructing distributed systems. The framework is capable of implementing large event-driven control systems for mission-critical facilities such as the NIF. Tools developed in this project were applied to the NIF example architecture in order to gain experience with a complex system and derive immediate benefits from this LDRD.

The ICCS integrates data acquisition and control hardware with a supervisory system, and reduces the amount of new coding and testing necessary by providing prebuilt components that can be reused and extended to accommodate specific additional requirements. The framework integrates control point hardware to a supervisory system by providing the services needed for distributed control such as database persistence, system startup and configuration, graphical user interface, status monitoring, event logging, scripting language, alert management, and access control.

At the heart of advanced distributed control system architectures is some form of interoperability services. Today, the most common services for object distribution are the COM/OLE component infrastructure and object bus by Microsoft, and CORBA which is a relatively new world-wide standard specified by the Object Management Group (OMG). Most people believe that CORBA is much better than COM/OLE, at least technically.

CORBA defines object interfaces with a standard language (IDL) then establishes transparent interoperability among objects developed with many kinds of languages and residing on a network of many kinds of distributed computers. The key word in this definition is ‘transparent’. Neither the Client nor the Server knows that they are separated by a network in a distributed system. This greatly simplifies the development of the application portion of the Client and Server given that the code that provides the extensive supporting services is well organized and tested, is not in the way of the application developer, and is yet still available for understanding and diagnostic purposes.

The even more recent popularity of the Java language, whereby the same language will operate unchanged on different kinds of computers, may in time reduce the overall learning curve for major projects. The performance cost of using an interpretive language such as Java will be offset over time by more powerful machines. At this date however, close integration of Java with CORBA is just developing and selection of Java for use in mission critical control system applications such as the NIF would be problematic.

Many products are available that build 'common' services for different languages including Java. Selection of the particular form of interoperability convention for a project is still quite controversial and sets the tone of future work, capabilities, and support for each particular project. Considerable effort is being expended by many companies in the marketplace to advance these opposing conventions.

Every kind of interoperability convention that adds support to the interaction between distributed objects suffers from additional complexity and slower speed of performance than can be obtained from traditional point-to-point communications. In the case of CORBA, a performance difference of roughly three-fold was established and is well recognized in an early paper on the subject by A. Gokhale and D. Schmidt.

However, the power / cost ratio of computers and networks continues to increase and is now well beyond the point where it is of primary concern for many systems. Comparisons however, with point-to-point communications (typically sockets via TCP/IP), continue to provide a useful performance metric for calibration of design and modeling efforts.

The ICCS architecture is based upon CORBA and is being implemented principally in the Ada95 computer language.

Designers of large, powerful, complex control system must actively deal with performance questions. For the ICCS, the completeness of suitable functionality, the speed of performance and utilization of machine and network resources, and the developing nature of the commercial CORBA products themselves, presented a certain risk. This LDRD thus evaluated CORBA in general, and a particular implementation, to determine its features, performance, and scaling properties, and to optimize its use within the ICCS. Both UNIX and real-time operating systems were studied. Awareness of CORBA products and services, and the status of world-wide developments, continues to guide software optimization as NIF software is being implemented and tested.

2. CORBA in the Literature

In this section the history of CORBA, competing paradigms, requirements of a Component, the Object Management Group, product organization and general services, the Interface Definition Language, the Client and Server side interfaces, the Object Adapter, the Interface Repository, initialization and connections, and some performance data are summarized from the views of several authors listed in the references.

Some of the references conflicted with each other, and we now see that some of the comments made by other authors were misunderstandings, in error, or were superceded by product developments that took other paths. Features of the product that we are currently using (and many others) have expanded over time but still do not complete all of the goals and expectations of CORBA, the OMG and world-wide user community.

With all of its capability and promised advantage, CORBA is a complex package of technologies and products which require quite a concentrated effort to master. Fortunately, we can successfully use a well-developed, well-tested, and robust subset of CORBA and its services for any particular system.

The current Ethernet era of client/server — which began about 10 years ago — is ending. Its being replaced by webs influenced by the exponential increase of low-cost bandwidth and a new generation of network-enabled multithreaded desktop operating systems with plug-and-play architecture. Millions of machines on the global 'information highway' can be both clients and servers.

The best way to think of CORBA is as the universal "software bus" where the connection between distributed objects is transparent and appears just as if the objects were locally connected.

CORBA provides a standard interface by which distributed objects can "plug and play" to interoperate with one another. Even when made by different vendors, at different times, the object interfaces are standard enough to coexist and interoperate. The interface types and methods between the Server Objects and the Client are defined using an industry standard Interface Definition Language (IDL). An IDL compiler provides all of the necessary interface code and templates into which use-specific statements are added.

2.1 Important Elements for Near Term Use

The important elements for our near term use are illustrated in the following figure.

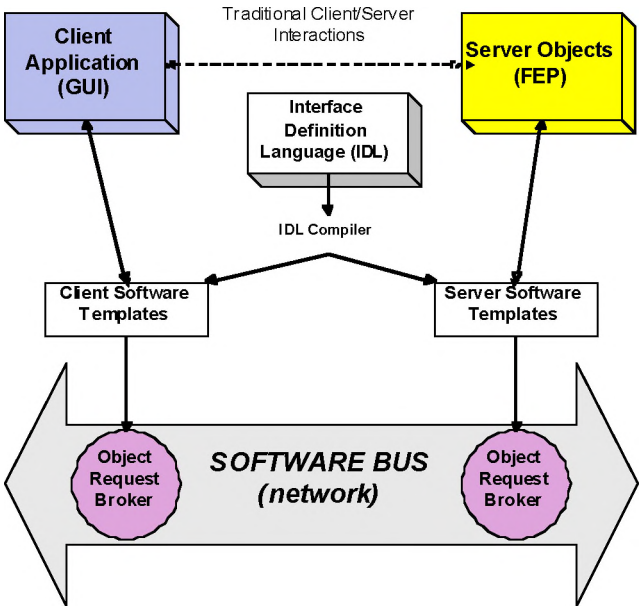


Figure 1 Important elements for our near term use

In summary of our literature search and prototype work, we decided that CORBA is the most likely candidate for the improved infrastructure required to implement large-scale, mission-critical, object-oriented, portable applications that will span palm-top to supercomputers over the next 5 to 10 years.

2.2 The History of CORBA

The design of CORBA was derived from the inputs of many people representing the various technologies, developments, and competing paradigms of the past. The 'Common' in CORBA stands for the combination of two original submittals to the OMG on its Request for Proposal — a static approach by Sun and HP and a dynamic approach by Digital and HyperDesk.

The following figure illustrates some of the historical developments that lead to CORBA.

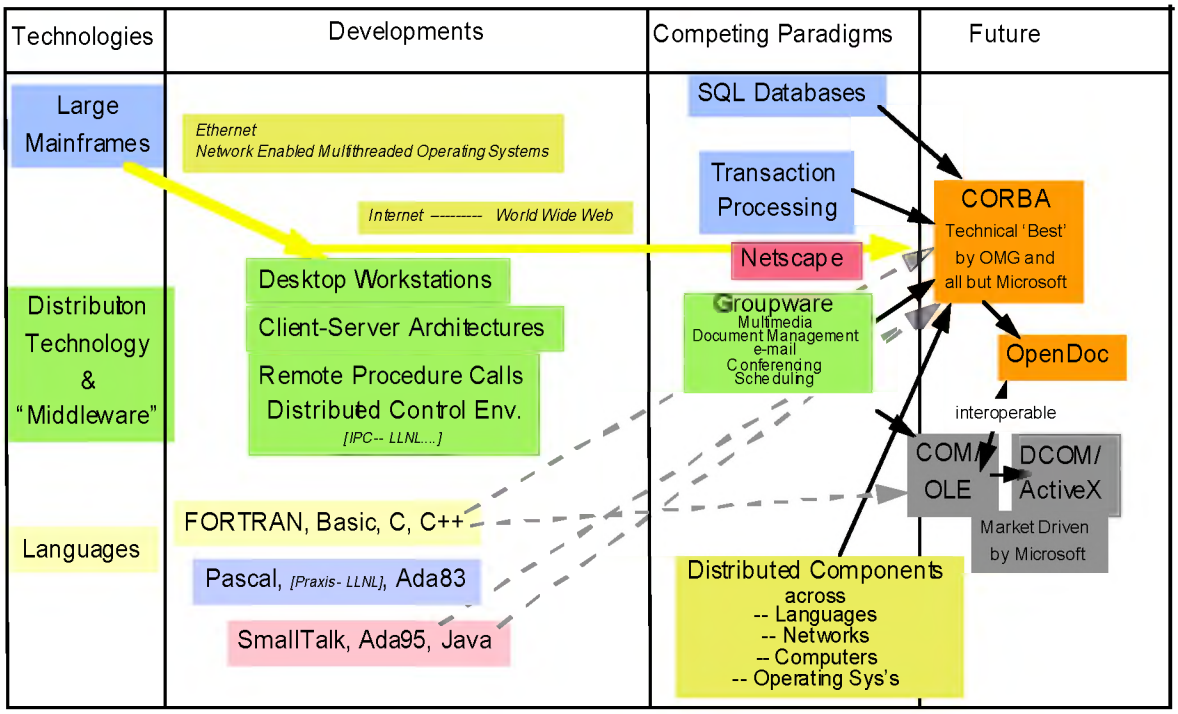


Figure 2 Historical developments that lead to CORBA

2.3 Competing Paradigms

An extensive literature review of competing paradigms for distributed interactions was conducted at a time when we had little experience with CORBA. There are four competing paradigms for developing new client/server applications:

- SQL (System Query Language) databases — by encapsulating SQL commands in named and compiled procedures that reside on the same server as the database (referred to as TP lite or stored procedures). SQL database servers are the dominate model for creating client/server applications today. SQL is poor at managing processes — hence the event of SQL front-end tools like Power Builder. Also, different vendors SQL based products do not interoperate well.
- TP (Transaction Processing) Monitors

  Transaction Processing Monitors have been used for many years on mainframes to manage processes and orchestrate programs by breaking complex applications into pieces called transactions and to make large applications act in unison to service thousands of clients. They are injected between the remote clients and the server resources in a middle tier, provide routing, load balancing, funneling (dealing with a subset of 1000's of clients at any particular time) and restart. They can manage transactional resources across multiple servers and can cooperate with other TP Monitors in federated

arrangements. The architecture is similar in many ways to that needed for the distributed object era.

TP Monitor vendors were slow to adapt to the Ethernet era and shrink-wrap market realities, and their products were overkill for the single-server/single-vendor, departmental sized applications that dominated the Ethernet era. TP Monitor vendors have a solid technical background to dominate in the distributed object era, and have been heavily involved in creating the CORBA standards.

- Groupware

  Groupware is a collection of technologies that allows representation of complex processes that center around collaborative human activities. It includes:

  - multimedia document management
  - workflow (automatic routing of events of work)
  - e-mail
  - conferencing
  - scheduling

  Groupware collects highly unstructured data — text, images, faxes, mail, and bulletin boards — and organizes it into a 'document' which can then be viewed, stored, replicated and routed anywhere on the network. The premier groupware product in industry is Lotus Notes.

- OpenDoc

  In recent years OpenDoc appears to be almost totally replaced by web browsers with Java capability. OpenDoc has been called "a CORBA object with desktop smarts". OpenDoc is object-oriented and 'network-distributed' to its core. It was designed to operate over wide area networks on completely different types of computers. It also has interoperability with Microsoft's OLE as a primary design goal. OLE objects can be embedded into OpenDoc documents.

  OpenDoc is important because it will be a primary way for CORBA to get to the desktop of non-Microsoft platforms (the other path is Netscape's support for the Internet InterOrb Protocol in future browers).

  OpenDoc documents can contain any number or variety of text, spreadsheets, video and audio clips, graphics, CAD/CAM drawings — even other compound documents. The fact that OpenDoc is designed to be independent of specific operating system is a critical difference with Microsoft's OLE/DCOM.

  The OpenDoc standard was developed by the Component Integration Laboratory — a non-profit industry association started in 1993 by Apple, IBM, SunSoft, Oracle, Novell, WordPerfect, Xerox, and Taligent. An independent certification program now exists to assure developers of compatibility with the standard. Certified parts are called 'Live Objects'.

- Distributed Objects or Components

  Components are objects designed and implemented by any of many methodologies, combined with other objects using distribution technology to become standalone distributed objects that provide the unit of work and distribution in a plug-and-play manner across networks, applications, languages, tools, and operating systems.

2.4 Requirements of a Component

Components are Objects (in the object-oriented design sense) with provisions and enhancements for distribution over many kinds of computers that can be operated with various languages, operating systems, windowing systems, networks, tools, and hardware platforms.

In principal, the important requirements of a Component are listed in the following table. Note that these are goals and that no present implementation of CORBA provides the total set.

- Self-contained — it is a self-contained, shrink-wrappable, marketable entity.
- Clean Interface — it has a well specified interface providing all necessary information for client use.
- Limited tasks — it performs a limited set of tasks.
- Unanticipated combinations — it can be combined in unanticipated ways with other Components to form a complete application.
- Extendible – it can be extended by inheritance from other Components and by polymorphism.
- Security – it protects itself and its resources, authenticates itself to its clients and vice versa. Keeps audit trails of its use.
- Licensing – it enforces licensing policies including per-usage metering.
- Versioning – it provides some form of version control and insures that its clients are using the expected version.
- Life cycle management – it must manage its creation, destruction, and archival; be able to clone itself, externalize its contents, and move from one location to another.
- Support for Open tool palettes — a component is imported within a standard tool palette to be assembled with other components using drag-and-drop and other visual assembly techniques.
- Event Notification – it must be able to notify interested Components when something of interest occurs.
- Configuration and Property Management – it provides an interface to configure its properties and scripts.

- Scripting — a component permits itself to be controlled via scripting languages by being self-describing and supporting late binding.
- Metadata and Introspection — provides information about itself upon request including a description of its interface, attributes, and methods it supports.
- Transaction Control and Locking — it transactionally protects its resources and cooperates with other Components to provide all or nothing integrity. It provides locks to serialize access to shared resources.
- Persistence — it must be able to save its state and later restore it.
- Relationships — it must be able to form dynamic or permanent associations with other Components and control other Components.
- Self Testing — it provides and runs its own diagnostics.
- Semantic Messaging — it must be able to interact with the vocabulary of the particular methods and domain–specific extensions it supports.
- Self Installing — it must be able to install itself and automatically register its factory with the Component Registry.

"Objects excite programmers (and geeks) who write software systems and applications for a living. Components excite users who have projects to finish as soon as yesterday, but who don't care about programming languages or protocols — they just want the components to be functional, fast, easy to use, seamless, and self-contained."

2.5 Object Management Group (OMG) and Contributors

There are two opposing methods of providing the distributed object bus and support that Components need to meet the above requirements. One method is Microsoft's approach with the COM/OLE and ActiveX component infrastructure and object bus. The other is the OMG's CORBA

Since 1989, a consortium of object vendors — The Object Management Group (OMG) — has been developing specifications for an open software bus on which object components written by different vendors can interoperate across computers, networks, and operating systems. Some of the principal contributors to the specification were: Expersoft, IBM, IONA, DEC, HP, HyperDesk, NCR, Novell, Object Designs, Sun Microsystems and SunSoft. The resulting core concept is CORBA. The final CORBA 2.0 specification was released by the OMB in March 1996.

There are many CORBA implementations on the market, some of which are listed in section 2.12. There are over 500 vendors working on CORBA-compliant software products. As in any product line there are various versions — the latest and best standard from OMG at the time this material was reviewed was CORBA 2.0, which these notes are based upon.

2.6 CORBA product Organization and General Services

Flexibility of overall system architecture while maintaining standard conventions is enhanced by using CORBA, since the object implementations can be configured to run locally and/or remotely without affecting their implementation or use.

The presence of object-oriented methodology in CORBA is the result of necessity, not of choice. CORBA supports the important aspects of object-oriented design (polymorphism, data encapsulation, and inheritance) between the boundaries of distributed objects.

The total CORBA architecture is illustrated in the following figure.

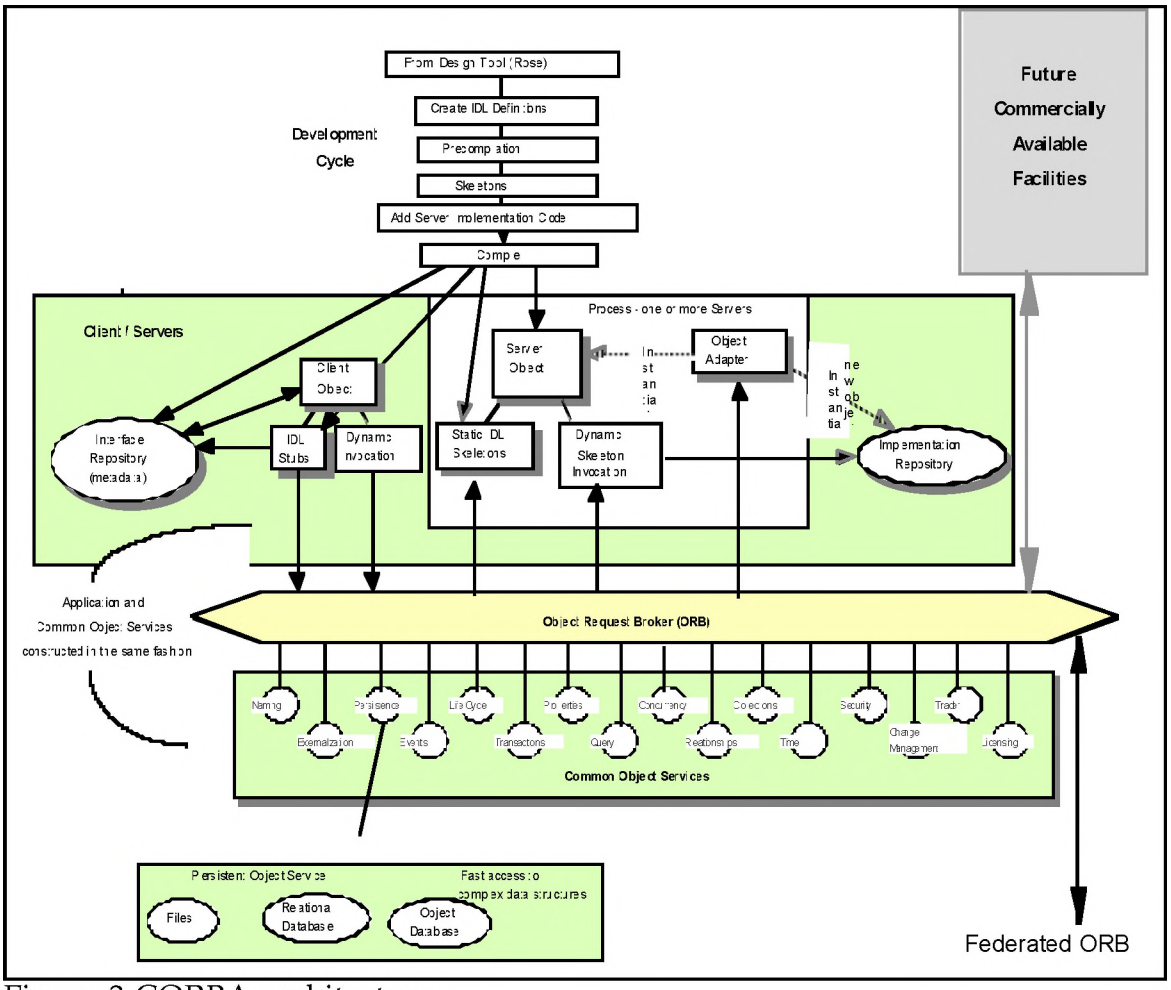


Figure 3 CORBA architecture

2.6.1 Object Bus

The object bus connects it all together. It supports nested transactions that span multiple servers, long-lived transactions that execute over long periods of time as they travel from server to server, queued transactions that can be used in secure business-to-business

transactions, roaming agents that look over business interests, active multimedia compound documents that can be moved, stored, viewed, and edited-in-place anywhere on the network, and sniffer agents which sit on the network at all times collecting information for system management, looking at trends, and gathering statistics.

2.6.2 Object Request Broker

The Object Request Broker (ORB) (also called an *object bus*) is the middleware that supports Component interoperations across machines, languages, operating systems, and networks. It makes different objects (and their associated data sets) reusable by different applications. The ORB intercepts calls of the Components, finds another Component that can implement the request, passes parameters, invokes its methods, and returns the result. Data encapsulation occurs because each client object knows little about the data it accesses when making requests of the respective objects through the ORB. Components exchange Metadata and discover each other by using services of the ORB. If one description of an object is designed to interface with the ORB, any object derived from that parent object will preserve its parent's interface. Note that the ORB itself is a self-describing Component whose description is contained within the Interface Repository (described below) and following the CORBA conventions.

A gateway infrastructure allows linkage of different ORBs. An integration path allows linkage between CORBA ORBs and other object-oriented approaches such as Microsoft's COM with OLE.

2.6.3 Object Services

In the design of any particular system, application objects are independently developed then CORBA services are mixed in by subclassing the original class and inheriting the services needed to make a Component. This is accomplished via IDL.

Object Services are provided by their own objects to support interactions between other objects. Note that not all of the important characteristics of Components (listed above) are supported yet in CORBA. In particular, there is currently no explicit support for real-time guarantees, recovery from partial failures, group communications, or causal ordering of events (see "Reliable Systems with CORBA" below). Currently an inconsistent set of available features is supported within the various implementations of CORBA. For example, although it receives great attention in the literature, the Dynamic Invocation Interface is not yet supported in the Objective Interface System ORBexpress (OIS)/Ada95 product used in the NIF control system.

Fortunately, many of the object services planned in the future are not required in any particular application and the important elements of it can be developed for that application. It seems that some of these planned services will lead to heavy machine loads and extra complexity for particular applications.

The object services include:

- Life Cycle — operations for creation, copying, moving, and deleting objects or groups of objects where groups are defined by the Relationship Service including containment and referential relationships.

  To create a new object, a client must find a factory object (meaning an object that is capable of instantiating an object of the desired type, allocating resources, obtaining an object reference, and registering the new object with the Object Adapter and Implementation Repository).
- Persistence — a single interface for storing components persistently.
- Naming — allows components to locate other components on the bus by name. Each named Component is a structure with an identifier string and a descriptive string such as a type definition. More than one name can be optionally associated with an object reference. Naming hierarchies can be created and clients can navigate through naming context trees in search of needed objects. To bind a name is to create a name-to-object association for a particular context. Names can be registered with the Properties Service allowing searches on properties such as time_last_modified, etc.
- Event Notification — allows components to dynamically register or unregister interest in specific events. Provides asynchronous interactions between anonymous objects (i.e. notification when things happen) using standard CORBA requests. Interactions are between suppliers and consumers through an event channel and can be either by pushing or pulling (aka polling). Features such as priorities, filtering, transaction protection, reception confirmation, time-to-live stamps, or queue management are important considerations.
- Concurrency Control — provides lock manager for transactions or threads.
- Transaction — provides two-phase commit coordination among recoverable objects using flat or nested transactions
- Relationship — creates dynamic associations (a.k.a. hyperlinks) between objects. Provides mechanisms for traversing the links that group objects. Can be used to enforce referential integrity constraints, track containment relationships, and other actions with any type of links between objects.
- Externalization — provides standard stream-like mechanism to get data in and out of an object. Mechanism is implemented in two steps: copy to stream, copy form stream to receiving object. And vice versa.
- Query — a superset of SQL for objects based upon the Object Query Language. Returns a collection of objects that satisfy the criteria specified via a select operation. Three interfaces provide operations on the result of a query:

  1. Collection Factory creates a new instance of an empty collection,

2. Collection defines operations to add, replace, retrieve, and remove members of a collection and insert at a particular location and create a movable pointer to navigate through the collection, and
3. Iterator provides operations to traverse a collection with Reset_to_start, Next_element, and a test for More_actions.

The Query Service also provides a Framework consisting of interfaces for dealing with the preparation and execution of a query.

- Licensing — meters the use of objects for charging to insure fair compensation. Provides a model for usage control of objects. Metering is per session, per node, per instance creation, and per site.
- Properties — dynamically associates named values or properties to any object.
- Trader (soon to be defined) — advertises object services and assists in finding them.
- Collections — (soon to be defined)
- Security — (soon to be defined)
- Time — (soon to be defined)
- Change Management — (soon to be defined)

2.6.4 Common Facilities

Support of complete systems is provided by common facilities. Common Facilities are collections of Components that provide services of direct use to application objects. There are two categories:

Horizontal Facilities

- User Interface Services — in-place editing services similar to those provided by OpenDoc and OLE.
- Information Management Services — compound document storage and data interchange facilities similar to those provided by OLE and OpenDOC.
- System Management Services — defines interfaces for managing, instrumenting, configuring, installing, operating, and repairing distributed object Components.
- Task Management Services — workflow, long transactions, agents, scripting rules, and e-mail.

Vertical Facilities

- Provide IDL–defined interfaces for vertical market segments such as health, retail, finance, and control systems.

2.6.5 Interface Definition Language (IDL)

The Interface Definition Language (IDL) allows an object to interact with the rest of the world by communicating that object's methods and parameters to other objects through the ORB.

IDL statements specify a components attributes, the parent classes it inherits from, the exceptions it raises, the type of events it emits, pragmas for generating globally unique identifiers, and the methods its interface supports — including input and output parameters and their data types. IDL is a subset of C++ with additional keywords to support distributed concepts; however, IDL does not include any procedural structures or variables. The goal of CORBA is to "IDL-ize" (sic) all client/server middleware and all components that live on an ORB. It is often used to encapsulate legacy software.

The metadata that describes the use of a component is generated automatically by an IDL compliant compiler or directly from an object-oriented language or tool.

Pragmas (special instructions to a compiler) are used to set a prefix that is appended to all indentifiers (ID), to set a version number, and to associate an arbitrary Repository ID with a specific IDL name.

2.6.6 Client Side Interface

2.6.6.1 Client IDL Stubs

The Client IDL Stubs provide a static interface to object services. The stubs act as a local call — a local proxy for a remote server object. A client has a stub for each interface it uses on the server. The stub includes code to perform marshaling (encoding/decoding the operation and its parameters into a flattened message that is sent/received to/from the server).

2.6.6.2 Dynamic Invocation Interface (DII)

A Dynamic Invocation Interface (DII) is also provided which allows objects to dynamically define at run time the information which would have been provided statically by an IDL. DII allows an application to issue requests on objects whose interface may not have been defined at the time the application was compiled.

Unlike IDL stubs which only allow Remote Procedure Call (RPC)-style requests, the DII also allows clients to make non-blocking deferred synchronous (separate send and receive operations) and one-way (send-only) calls.

We are not currently using the DII in the NIF control system but need to be aware of it for possible uses in the future.

2.6.6.3 Interface Repository Application Programming Interface

Provides services to obtain and modify the descriptions (metadata) of all registered Component interfaces, the methods they support, and the parameters they require.

2.6.6.4 ORB Interface

A direct interface to the ORB provides a few local services such as conversions from object reference to name string, and some language and system bindings to handle differences of specific implementations.

2.6.7 Object Adapter

The Object Adapter is how a CORBA object knows of and uses the services of an ORB.

2.6.7.1 Server Side Interface

2.6.7.2 Server IDL Stubs or Skeletons

Provides static IDL interfaces (*Skeletons*) to each service exported by the server.

2.6.7.3 Object Adapter

The Object Adapter contains the primary mechanisms for an object implementation to access ORB services and provides the environment for running the server application. The Object Adapter sits on top of the ORB's core communication services and accepts requests for service on behalf of the servers objects. Actions include:

- Registers server object implementation classes with the Implementation Repository.
- Broadcasts the services it provides on the ORB, and responds to directory type queries.
- Instantiates and activates new server objects at run time. The number of instances created is balanced as a function of the incoming client traffic load.
- Authenticates Client making the call. Security actions are left to the specific implementation.
- Generates and manages object references. Assigns references (unique IDs) to new objects and maps between implementation-specific and ORB-specific representations of object references.
- Processes incoming client calls. Peels off requests, and transfers it to the interface stub which interprets the request and incoming parameters and presents them to the object's method invocation.

The CORBA specified standard Object Adapter is called the *Basic Object Adapter*.

CORBA defines four object activation policies:

1. Shared server — multiple objects reside in the same server process (program)
2. Unshared server — each object resides in a different server process (program)
3. Server-per-method — a new server is started for each request

4. Persistent server — servers are activated by means outside of the Basic Object Adapter

2.6.7.4 Implementation Repository

A run-time repository of information about the classes a server supports, the objects that are instantiated, and the ID's. Serves as a common place to store additional information associated with implementation, trace information, audit trails, security, and other administrative data.

2.6.7.5 ORB Interface

(Identical to that provided on the Client side)

2.6.7.6 Dynamic Skeleton Interface (DSI)

The server side of the Dynamic Invocation Interface (DII) provides run-time binding and is used by the ORB to issue incoming method calls to objects that are implemented independently and do not have IDL-based compiled skeletons or compile-time knowledge of the implementations. They can be used by interpreters and scripting languages to dynamically generate object implementations.

But note that static interfaces are easier to program, provide more robust type checking, provide better performance, and are self-documenting in that one can tell what occurs by directly reading the code.

2.6.8 Interface Repository

Interface Repository — is a run-time database that contains dynamic metadata, machine readable (compiled) versions of the IDLs for the objects known by the ORB. These definitions may be captured directly from an IDL-compiler or through the Interface Repository write functions. Interface Repositories can be maintained locally or managed elsewhere. An ORB may access multiple Interface Repositories. An interface (or entry in the Interface Repository) is defined for each of the 8 IDL structures:

1. Moduledef — defines a logical grouping of interfaces as a module.
2. InterfaceDef — defines the objects interface, contains lists of constants, typedefs, exceptions, and interface definitions.
3. OperationDef — defines a method at an objects interface, contains lists of parameters and exceptions raised.
4. ParameterDef — defines an argument of a method.
5. AttributeDef — defines the attributes of an interface.
6. ConstantDef — defines a named constant.
7. ExceptionDef — defines an exception that can be raised by an operation.

8. TypeDef — defines the named types that are part of an IDL definition.

Each Interface Repository is represented by global root object, which contains the 8 possible IDL structures in a hierarchy illustrated in the following figure.

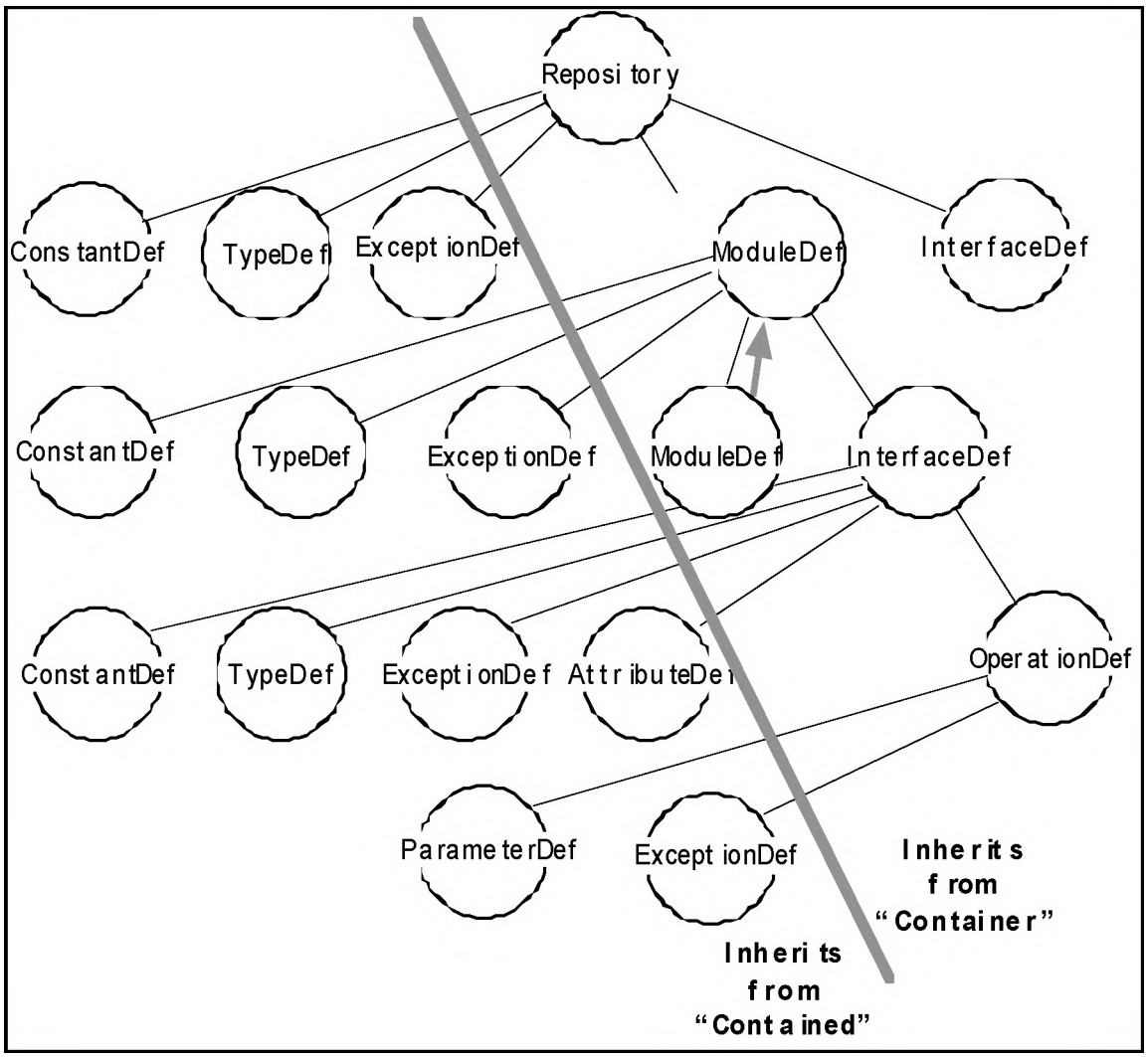


Figure 4 Containment & Inheritance Hierarchy for the Interface Repository Classes

Navigation and extraction of information from the Interface Repository is accomplished with nine methods:

1. Describe — returns a Description structure containing the IDL that describes a contained object.
2. Lookup — returns a sequence of pointers to objects within a contained object.
3. Lookup_name — locate a named object within a contained object.
4. Contents — returns a list of objects contained or inherited by the contained object.

5. Describe_contents — returns a sequence of pointers to the content descriptions of objects within the contained object (a combination of Describe and Contents).
6. Describe_interface — returns a structure describing an InterfaceDef object.
7. Is_a — returns TRUE if contained InterfaceDef is identical to or inherits directly from an interface specified in an input parameter.
8. Lookup_ID — look up an object in a Repository.
9. Get_primitive — obtain a reference to a primitive object of a Repository such as base data types and codes.

2.7 Initialization and Connections

A Component in a CORBA compliant system performs the following initialization steps to bootstrap itself into operation:

- Informs ORB of presence via a CORBA Application Programming Interface (API) call to *ORB_Init* and obtain a reference to the ORB Component.
- Inform Basic Object Adaptor (BOA) of presence via invoking method *BOA_Init* within the ORB and to obtain its object reference.
- Invoke method *List_Initial_Services* to obtain a list of well-known objects, for example the Interface Repository and Naming Services.
- Invoke method *Resolve_Initial_References* to obtain references for the services required.

2.8 Interconnections of ORB's

The General Inter-ORB Protocol (GIOP) specifies a set of seven message formats that cover all the ORB request/reply semantics to provide ORB-to-ORB interconnections over any transport protocol. A Common Data Representation (CDR) maps data types defined in the OMG IDL into a flat message representation.

CORBA 2.0 specified the mandatory Internet Inter-ORB Protocol (IIOP) which defined how GIOP messages are exchanged using TCP/IP connections, making it possible to use the Internet itself as a backbone ORB.

For application specific networks, Environment-Specific Inter-ORB Protocols (ESIOPs) are specified. The first ESIOP uses Distributed Computing Environment (DCE) where IDL and CDR types are mapped directly into DCE's native Network Data Representation (NDR). The DCE ESIOP provides a robust environment for mission-critical ORBs with a rich set of features including efficient large data transfers, Kerberos security, cell and global directories, distributed time, and authentication. Some of the major features within DCE such as Remote Procedure Calls are not currently applicable to CORBA and hence, even though very robust, represent unneeded overhead. DCE is supported by IBM, Digital, Tandem, and HP who will support it on their ORBs.

Microsoft is attempting to create COM directly on top of DCE for world-wide (but proprietary) use.

2.9 Performance of CORBA in the Literature

The speed of performance of CORBA has been measured as about 1/3 as fast as 'the best C++' code for small data packets (1000 bytes) between two dual-processor SPARCStation Model 712's connected with an ATM network. The top speed of simple non-structure data transfer of large blocks (100K bytes) is about the same for C++ code and CORBA (50 Mbytes per second). Reference 8 contains many plots of performance of Orbix and ORBeline CORBA versus C++ versions of TTCP and RPC for buffer sizes from 1000 bytes to 140 Kbytes. Experience of the authors indicates that present CORBA implementations are well-suited for request/response applications over lower-speed networks (such as Ethernet) however considerable overhead is obvious within CORBA when higher speed networks are used.

Overhead comes from a variety of sources:

1. non-optimized conversions, data copying, and memory management
2. generation of non-word boundary aligned data structures by the CORBA stub compilers
3. excessive control information carried in request messages
4. inefficient and inflexible receiver-side demultiplexing and dispatching operations
5. long chains of intra-ORB function calls
6. lack of integration with the underlying operating system mechanisms

(Items 1, 2, 3 and 6 are probably the areas that we need to be aware of for large distributed control systems.)

The latency for sending richly-typed data increases rapidly as the buffer size increases with Orbix. Two-way transfers of richly-typed structures of 1000 bytes are reported in the literature as having a latency of about 45 ms on the test platforms being used. Measurements of total transaction time for the LDRD simulation studies were significantly less, in the order of 2 ms., which is probably due to the use of more powerful processors.

Numerous suggestions are made in the referenced papers for optimizing CORBA including revisions of the specifications. It appears that this work is ongoing and could well result in optimized versions of CORBA and better interfaces with real-time operating systems.

2.10 Reliable Systems with CORBA

Neither the CORBA standard nor conventional implementations of CORBA directly address complex problems related to distributed computing such as real-time, high-speed,

quality of service, partial failures, group communications, and causal ordering of events. The CORBA model itself does not provide solutions to the problems of detecting, and reacting, to partial failures, and there is need for distributed debugging tools and run-time validation tools. The reference proposes extending the ORB (using object-oriented extensions) to add appropriate lessons learned from other models to increase the reliability and availability of CORBA-based interactions and systems. Again, it appears that this work is ongoing and could well result in improved versions of CORBA before needed on the NIF.

2.11 Real Time Operating System Interface with CORBA

The performance of current CORBA implementations is not suitable for latency-sensitive real-time applications, including real-time systems (e.g. avionics), and constrained latency systems (e.g., teleconferencing). The interface between CORBA implementations and underlying operating systems is inefficient in many respects. Areas identified for improvement include: resource scheduling mechanisms, multiplexing, data copying, and byte-order alignment conversions (which often occur at several layers), and the lack of ability to select between using compiled code versus interpreted code for conversions.

2.12 Suppliers and Users of CORBA

Vendors are shipping CORBA-compliant ORBs on all major operating systems from the Apple Macintosh OS, Windows (all flavors), more than 20 different UNIX operating systems, to Digital's OpenVMS and IBM's MVS.

Identified vendors and CORBA-compliant ORB products include:

- BBN's (Corbus)
- Component Integration Laboratories (OpenDoc)
- Chorus Systems (CHORUS/COOL ORB)
- Digital Equipment Corporation's (Object Broker)
- DNS Technologies (SmalltalkBroker)
- Expersoft (PowerBroker CORBAplus)
- Hewlet Packard (ORBplus)
- Iona (Orbix)
- IBM (Distributed System Object Model —DSOM)
- Object Interface Systems (ORBexpress/Ada) (being used on NIF)
- Object-Oriented Technology (Distributed Object Management Environment — DOME)
- O/SPACE (CORBA implementation for Java)
- Sun Microsystem (NEO)
- Tandem (nonStop DOM)

- TRW (Universal Network Architecture Service — UNAS)
- Xerox PARC (ILU)

Recent announcements of important users, applications, and tools include:

- Chevron has announced one of the largest examples to date of plans to use CORBA technology to link engineers' desktop systems with browsers to geographical, seismic, and historical drilling information stored in databases on many kinds of computer platforms all over the world. They have selected the IONA CORBA implementation. (Computer World, April 28, 1997)
- The Gap has 1900 stores around the world and posted $5.3 billion in sales in 1996. Keeping everyone connected to the most up-to-date information on products and sales is important. They use IBM mainframes at the backend, Sun Solaris servers at the middle tier, and OS/2 and Windows NT- based desktops. A corporate decision was made to go to object-oriented technology and to access information with browser technology by using CORBA with Java. They have selected the Visigenic's CORBA software. "Putting this object-oriented system together is a big job — a lot of pieces and ways to go wrong" (Phil Wilkerson, The Gap Inc., Computer World, June 9, 1997) The switch over to the new system is expected in 6-8 months.
- The cover story in the January 1999 Component Strategies magazine "Building Large Scale CORBA-Based Systems."

3. The CORBA Test Package

With all of its capability and promised advantage, CORBA is a complex package of technologies and products which requires quite a concentrated effort to master.

We developed three different sets of test software to insure that CORBA, combined with the Ada95 programming language, would be viable tools for large distributed control systems including the NIF, and to address risks and concerns raised at project reviews. The approaches were:

- An extensive "CORBA Test Package" which tests all kinds of CORBA interactions between multiple processes with multiple objects on multiple machines.
- Small special case programs for measuring the optimum speed of performance.
- A simple Java language Client was constructed to interact with a Server from the CORBA Test Package (written in the Ada95 language). This was to illustrate interoperability between commercial CORBA products.

The CORBA Test Package fully tests the CORBA 'middleware' and associated Ada functions, independently from the ICCS control system design, yet using all the features of CORBA used in that design. Regression tests can be performed to compare the functional capabilities and speed of performance between various versions of the

operating system, network hardware, compiler, configuration management system, and the commercial CORBA software (ORBexpress from OIS).

The CORBA Test Package was used extensively during early use of the commercial products to test (a) functionality with a large number of objects and distributed servers, and (b) speed of performance. Errors and misunderstandings were reported to the supplier and resolved with subsequent releases.

An extensive series of tests of all kinds of interactions between multiple objects on multiple computers was conducted. Many interactions with the technical developers of the products at the companies involved were organized through the testing process, and many problems were found and resolved.

These efforts led to the current state of affairs where the CORBA and Ada products selected for the NIF are now successful, well established and being used, and the center of concentration is on implementation of the applications themselves. Speed of performance of present products appears to be well able to handle the NIF control system requirements as presently understood. We may well find the need to resume the concentrated attention to CORBA related to configuration and diagnostic capabilities as deployment approaches.

3.1 Structure and Capabilities of the CORBA Test Package

The structure and capabilities of the CORBA Test Package are categorized as follows.

Client — The Client package is capable of transactions with any number of Objects (1000 used so far) on any number of Servers (12 used so far) on any number of computers (4 used so far). It measures and records process and clock time, and memory usage of both the Client and Server(s).

Servers — Any number of copies of the Server can be operated from the Client on any of the computers in our network with CORBA installed. To date, 12 Servers have been operated concurrently on 4 different machines. Operations can be in serial or parallel (i.e. the Client is multi-threaded, one or more threads per Server).

Objects — The Servers consist of an 'initial object' that has both simple methods and 'object factory' capability. Any number of subsequent objects can be created (with the object factory), used, and deleted by the initial object of the Servers. To date, the creation of up to 1000 objects per Server has been tested. I/O operations are performed directly to the initial object created for each Server and to each subsequent Object created on each Server. A small number of very large Objects (1,000,000 bytes) can be created in each Server. These creations dynamically allocate memory, which is not part of the built-in object data thereby testing this capability for other applications.

Data Transaction Tests — All types of standard CORBA data types including fixed and flexible length arrays, structures, and the 'any' type are tested. Sequenced and Unbounded Strings have been tested up to a length of 10,000 bytes.

Write, Read, and Write/Read operations can be selected. Unique data is written to each Object upon each type of write operation. When Reads are performed on objects which have previously been written to, an error check is made to insure that data read from the object is the same as was written for each type of data.

Exceptions — Upon command from the Client, exceptions can be raised through the IDL interface from both the initial Servers object and from subsequently created Objects within the Servers.

Call backs — The Server 'initial object' can perform call back operations from the Server to the Client that are asynchronous with data transactions and other functional tests.

Binding to Objects — CORBA References for each created Object are kept in the Client, but optionally, the Client will erase existing references and perform a Bind operation to reestablish connection with Objects using the previously established names of each Object. This capability is part of the OIS ORBexpress design but is not utilized in the ICCS control system except to establish an initial link at system start up.

Multi-tasking — Both the Client and Server have multi threaded capability and can operate with any number of concurrent tasks to test performance / memory tradeoffs.

Timing Loops — Loops of any duration can be performed on the methods on the initial Servers object and on each object subsequently created within the Servers. A maximum time to spend in the loop of any particular test can be set. So for example, one can set each type of test to be performed for a given time rather than a given number of loops.

Performance Measurements — Clock time, processor time, stack size, heap size, and image size are measured for the Client and Servers for each kind of I/O operation, for raising exceptions, and for the create, delete, and optional bind operations. A file is written containing the conditions and results of each use. Reported times are in microseconds per successful loop of each kind of selected operation. Performance measurements on many combinations of these operations have been made and reported.

Diagnostics — Switches can be set to turn various levels of trace messages on/off.

CORBA versus Socket level testing: Many of the CORBA test operations also have an equivalent socket level test capability to establish a functional comparison and performance baseline upon which to judge CORBA.

Test Instructions — The performance of the Client and Server(s) is controlled by an instruction file, which is read upon startup.

3.2 CORBA Test Package Instruction File

An Instruction file is used to direct the Test Package Client on the specific tests to be performed. Fields in the Instruction File set test parameters for the following purposes:

3.2.1 General Control Parameters:

- Enable CORBA Testing

- Enable Socket Testing (socket tests are also built in to compare results)
- Set number of Client Tasks (if there are not as many client tasks as Servers, transactions to some of the Servers will wait until the others are complete)
- Enable Server Callbacks
- Enable Server Explanations (trace and error messages)
  - Enable Server Entries trace messages
  - Enable Server Data entry messages
- Set number of Servers Used
- Set number of Concurrent Servers
- List of Server Host Names
- List of type of machine for each Server
- Set Size of Client Tasks — sets amount of memory allocated for each Client task. Must be sufficient to store all the data related to a particular Server.

3.2.2 Parameters for testing each Server

- Enables Null OP — the simplest transaction
- Simple Variable I/O — sends a floating point number to / from the Server
- Consume processor time — causes the Server to consume extra processor time
- One Way Operations — tests one way transactions to the Server

3.2.3 Parameters for testing each Objects within each Server

- Number of Objects per Server — each Server creates this many objects for testing
- Number of Objects with Huge Arrays — causes each Server to create this many objects with huge arrays (1,000,000 bytes) (use sparingly)
- Simple Variable — send floating point number to/from each Object
- Little Array — send array of 100 octets (bytes) to/from each Object
- Big Array — send array of 10,000 octets (bytes) to/from each Object
- Little Float Array — send array of 100 floats to/from Objects
- Big Float Array — sends array of 10,000 floats to/from Objects
- Structure — sends a simple CORBA structure to/from Objects
- CORBA Any — sends simple CORBA 'any' to/from Objects
- The following parameters send unbounded strings to/from Objects. Parameters define initial size, and update size for each pass. This is handy for measuring transactions of varying size.
  - Size of Unbounded Strings
  - Add to size of Unbounded Strings on each pass

    - Multiply size of Unbounded Strings on each pass
- Size of Sequences of Longs — Send a sequence of 1000 longs to/from Objects
- The following parameters send Octet arrays to/from Objects. Parameters define initial size, and update size for each pass. This is handy for measuring transactions of varying size.
    - Size of Sequences of Octets
    - Add to size of Sequences of Octets on each pass
    - Multiply Size Sequences Octets each pass
- The following parameters send unbounded float arrays to/from Objects. Parameters define initial size, and update size for each pass. This is handy for measuring transactions of varying size.
    - Size of Sequences of Floats
    - Add to size of Sequences of Floats on each pass
    - Multiply Size Sequences Floats each pass

3.2.4 Parameters for testing operations between Clients and Servers

- Enable test of Bind Operations — test CORBA bind operation to Objects by deleting references to objects and reacquiring them with the Bind function.
- Enables Exceptions from server — causes CORBA exceptions to be raised from Servers and caught by the Client.
- Enables Exceptions from Objects — causes CORBA exceptions to be raised from Objects and caught by the Client.
- Delete Objects when completed — deletes Objects from Servers upon completion of a test run so that when the next test is begun the Objects are recreated.

3.2.5 Parameters for generating reports

- Enable Statistics — turns statistics reporting on/off
- Report errors — turns error reporting on / off. Errors are detected by comparing data returned from each object to data sent to each object, and printing cases which do not agree
- Number of Loops for Timing — sets number of loops of each individual test functions. Set to 100 for short development, about 10,000 for valid average timing tests.
- Record all times of Last Sequence Octets — used to detect a problem with a previous version of the product where occasional very long transaction times occurred
- Loop forever on Sequence Octets Test

- Max Duration of Each Loop (in Seconds) — sets maximum time allowed to complete a single test function. This allows the tester to go home at a 'reasonable' time. Statistics recorded about average times are still valid as the actual number of completed transactions is used rather than the instructed number.
- Delay between loops (in Seconds)— sets delay between major test loops
- Number of Total Cycles of Test Package — sets number of major test loop cycles

3.3 Tests on Solaris

Multiple Servers can be operated on any Solaris-based computer including the one that the Client is operated on.

3.4 Tests on VxWorks

VxWorks testing is immature. Many types of data transactions have not yet been tested. Servers have been tested on three VxWorks-based computers with varying results probably because the processor and network topology were changing in the test equipment area for other reasons.

4. Functionality and Performance Measurements

Many different tests of functionality and measurements of performance for simulation studies have been made of CORBA on various computers and network configurations, and with various versions of the CORBA product and language compilers being used. Some of the results follow.

4.1 Comparison of Speed of various CORBA types

As input to the design of the NIF control system architecture, we measured the relative speed of transactions of all CORBA types as provided by the IDL on all versions of the product on various machines. Typical comparative results (on medium powered processors) are illustrated in the following figure.

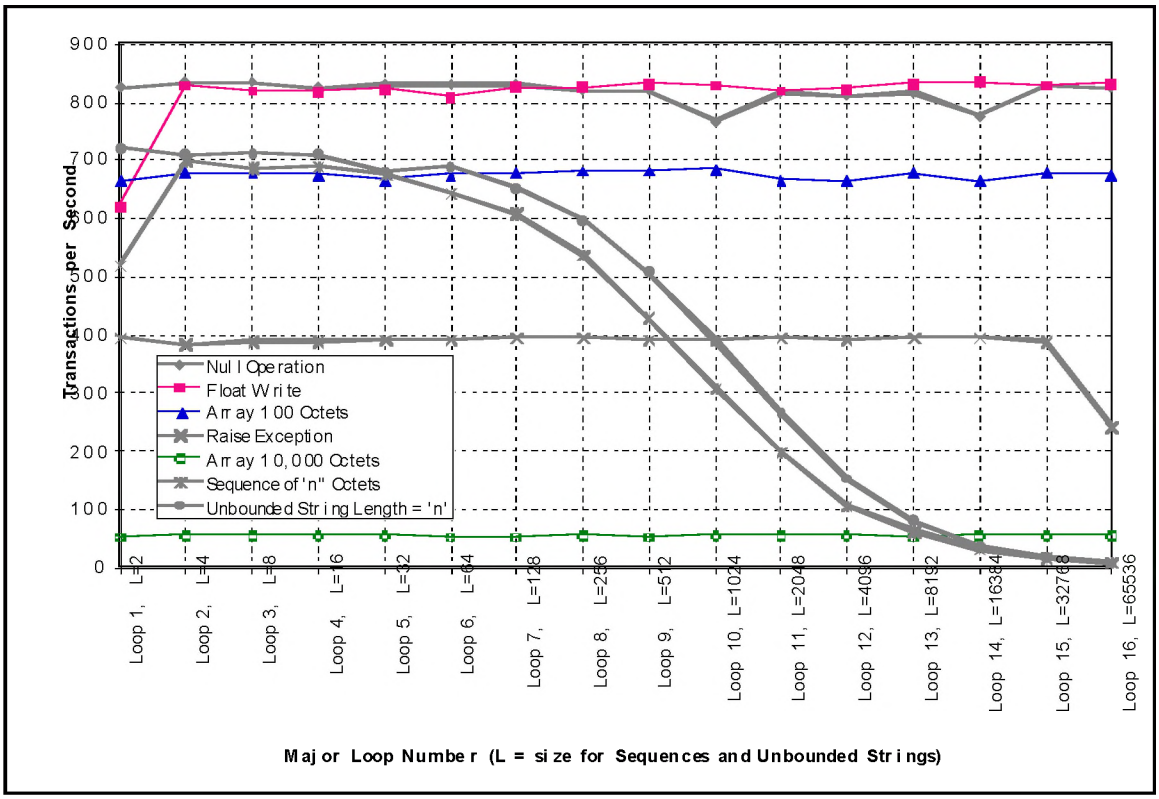


Figure 5 Comparative results of speed of various CORBA types

The X axis is the major loop counter through the program. Sequences and unbounded array sizes are changed for each major loop, but the other types remain the same. Plots of these other types give some indication of the variability of performance data (as described above). Note that, at least for this and all previous versions of CORBA, unbounded strings are almost always the fastest method of transmitting data types where the length is not known until run time (i.e., dynamic lengths). The anomalies at the beginning and end of some curves may be due to long start up times, memory size problems at the end, or a strange behavior of the particular version of ORBexpress being used at that time periodically pausing. This problem has since been corrected.

4.2 CPU Utilization:

CORBA uses considerably more processor time than socket based communication. Tests indicate that for message sizes below about 500 bytes in length the dominate factor in speed is the power of the processors on each end of the transaction, not the network connecting the processors.

Measurement of CPU utilization has been somewhat illusive. Measurements of process time need to be combined with information on the number of multiple processors installed on each particular computer to get the percentage of that machine used during elapsed wall clock time. Some previously reported results on multiple processors have inconsistently factored in this parameter.

4.3 Predictions of Performance within the NIF

In order to show that the delays and consumption of machine resources by CORBA within the ICCS architecture were affordable, we picked a few 'extreme' cases of the use of CORBA within the established NIF shot timeline. The selected cases were:

- Case 1: The entire automatic alignment of NIF based upon the "Simulation of Beamline Alignment Operations" by Mark G. Miller and Cynthia (UCRL-ID-tbd).
- Case 2: The busiest time slice during the countdown immediately prior to a shot, based upon the NIF Interface Control Documents, which give expected network traffic.

4.3.1 Case 1: Beamline Automatic Alignment Operations

Measurements of the clock time required for typical CORBA transactions, combined with the kinds and quantities of operations planned for within the automatic alignment model, indicate that CORBA based transactions with the alignment front-end processors will use no more than 2% of the available time on the planned processors.

The total number of CORBA based commands for this case is:

Video requests	17,000
Gimbals adjusted	8,000
Device setup	19,000
Alerts, Status to Supervisory	4,000
Total	48,000

The required maximum total alignment time is one hour. Forty minutes are allowed for manual operation, leaving 20 minutes for automatic operations. Two ms per CORBA message is a very conservative estimate of CPU utilization. Four processors are used in a load sharing arrangement.

2 ms x 48,000 messages / 4 processors = 2% of the allowed 20 minutes.

In this case, since the CORBA messages are small, and are divided between many different machines on a switched network, network consumption is minimal.

The ratio of time spent in each of the functions within the automatic alignment is illustrated in the following diagram:

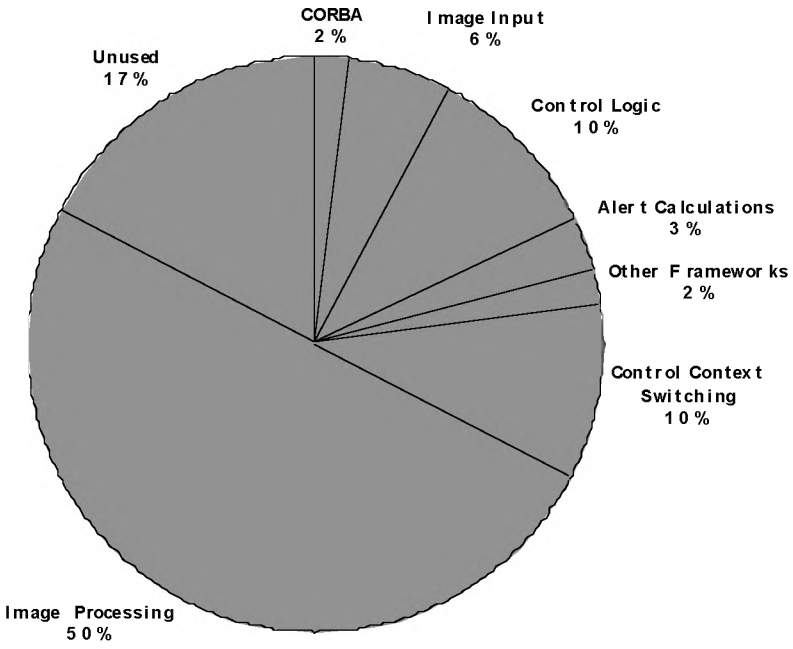


Figure 6 Automatic Alignment CPU Utilization per Processor

4.3.2 Case 2: Busiest time during shot countdown

The principal activities leading up to a NIF shot are illustrated in the following figure.

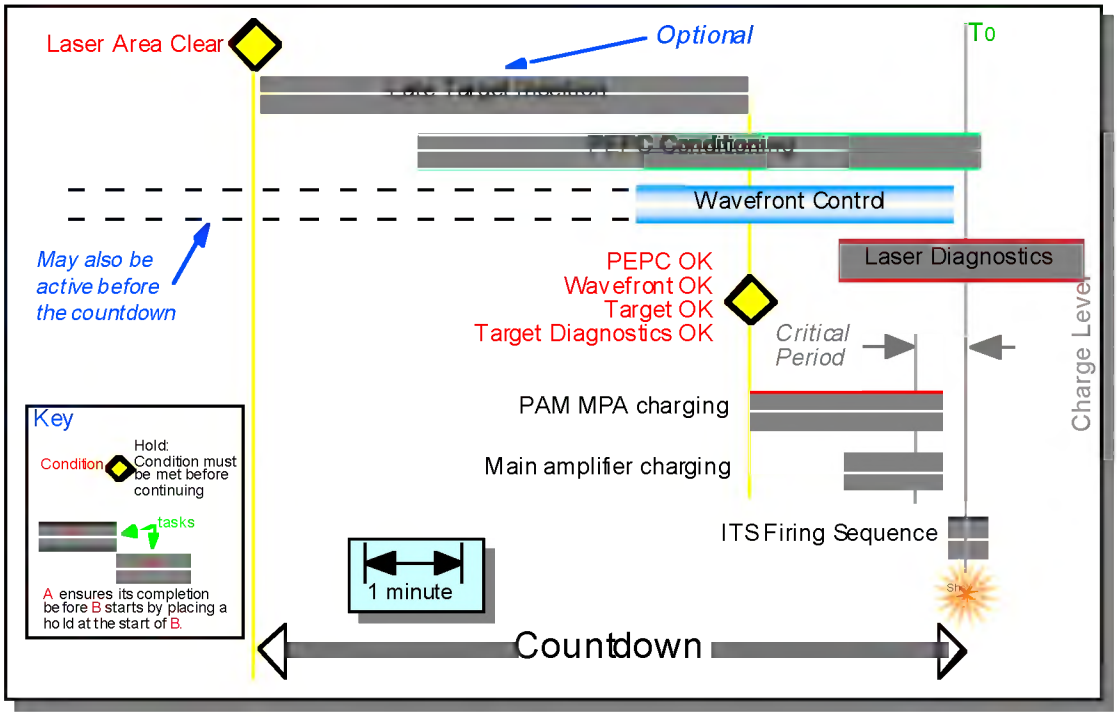


Figure 7 Principal activities leading up to a NIF shot

Review of the NIF Interface Control Documents (ICDs), which give planned network traffic indicate that the estimates given are possible (or requested maximum burst rates) but are not reasonable longer-term (i.e., minutes) or 'normal condition' rates. They also include traffic for diagnostic purposes that would not normally occur during a functioning shot. Short-term burst requests will simply be spread out over time in an event-driven system such as NIF. Unfortunately, the ICDs did not show the estimated total number of commands or read operations required to perform a function. Such information within the alignment model is easy to extend to various other purposes. Discussions with some of those responsible for the ICDs indicated that realistic message traffic during this period will not be as extensive as that expected during the automatic alignment case (above). However, it was concluded that further review of Case 2 should be conducted after better definition (or actual implementation and testing of prototype software) at the application, framework, and FEP levels.

Related aspects of this case are fully covered in the companion LDRD report "Countdown Status Messages Simulation" (UCRL-ID-133242).

4.4 Conclusions on Performance Measurements

The speed of performance of current versions of CORBA on current machines are reasonably well known and can readily be measured for subsequent versions, other conditions matching future ICCS software design, and on newer machines and network configurations.

Many possible optimizations of the use of CORBA, and improvements to the test package and measurements methods, have been identified. The effort of testing the performance of CORBA has identified methods in software and hardware for testing the planned production prototypes of the ICCS.

Any good prediction of performance of CORBA within a large distributed control system depends upon implementation and measurement of prototype versions of the system frameworks, supervisory applications, and front-end processors. Particular implementations set the level, style, quantity, and frequency of CORBA transactions.

All information available to date indicates that the necessary CORBA portion of the speed of performance of the ICCS can be readily obtained with:

- the power of the processors and network being designed into the system,
- the continued close cooperation of the staff at OIS,
- the many opportunities for optimization of software with additional effort as implementation proceeds and problem areas are detected, and
- the availability of even more powerful processors, the future.

4.4.1 Recent Performance Measurements

Due to limited resources and the need to rapidly recompile and test software to find performance problems, recent performance measurements were conducted with several versions of CORBA with the separate smaller software package mentioned above. The results of these tests are reported directly in the LDRD Final Summary Report and have not yet been replicated in the CORBA Test Package described herein.

5. Lessons from Current Publications Continue

The literature is now full of articles about using CORBA in various applications. One of the most relevant recent articles was the cover story in the January 1999 Component Strategies magazine "Building Large Scale CORBA-Based Systems" (ref. 13).

In this article, the authors point out numerous aspects of the use of CORBA and lessons learned in other systems (large telecommunication and business systems) that may be valuable when compared to our designs and current implementation. They describe things as confidentiality of data and protection from unauthorized changes, CORBA services that are now available with some commercial products. They express a strong belief that fine-grained CORBA objects (which we are using) can lead to problems of scale. They describe lessons learned from the use of distribution adapters, the Dynamic Skeleton

Interface, bulk and group operations and other techniques to exploit economy of scale, and handling connection failures including monitoring errors. They mention techniques related to robust memory management.

Recently the principal author was invited to visit LLNL and spoke with our development team for a day. These discussions lead to observations about commercial products in general, and several suggestions and observations of shortcomings about our current design and prototype implementation, which are now being incorporated and corrected.

Earlier CORBA papers discussed sniffer agents which sit on the network at all times collecting information for system management, looking at trends, gathering statistics. In the deployment phase of large distributed control systems, the ability to monitor control system performance and to actively cause errors at various layers of the hardware and software to test responses will be important. Insight into how and where such things have been used would be valuable.

Awareness of CORBA products and services, and the status of world-wide developments, must continue to guide software implementation and optimization of any large distributed control system as it is being implemented and tested.

Acknowledgements

The principal investigator wishes to acknowledge the work of collaborators drawn from several divisions across the Laboratory. E. A. Stout solved many detailed problems in development and testing, and developed the small independent test programs recently used for speed of performance tests. M. Komsthoeft, a summer student, developed the Java Client to illustrate connectivity between commercial products and languages.

References

Itemized references to phrases are not generally shown within, in the interest of avoiding errors of who said what first. However, the work and publications of many people including the following were valuable and appreciated.

1. "The Essential Distributed Objects Survival Guide", R. Orfali, et.al. Wiley, 1996.
2. "Programming with CORBA", Gabriel Minton, Unix Review, April 1996.
3. "Notes on OLE and CORBA", D Katiyar, Datiyar@eng.sun.com
4. "The Common Object Request Broker", v2.0, OMG.
5. "CORBA Services", 3/28/96 OMG.
6. "Common Facilities Architecture", 11/95 OMG.
7. "Measuring the Performance of Communications Middleware on High-Speed Networks", A Gokhale and D Schmidt, Dept. of Computer Science, Washington University, St. Louis, MO (http://www.cs.wustl.edu/~schmidt/).

8. "Constructing Reliable Distributed Communications Systems with CORBA", S. Maffeis and D Schmidt. to be publilshed in IEEE Communications Magazine, Vol 14, No 2, February 1997.

9. "Operating System Support for High-Performance, Real-Time CORBA", A Gokhole, D. Schmidt, T. Harrison, G Parulkar, Proceedings of the 5th International Workshop on Object-Orientation in Operating Systems, October 1996, Seattle, Washington.

10. "Collaborating Distributed Objects Over the Web", Rich Lysakowski, Scientific Computing & Automation, August 1996.

11. "OpenDoc and the Move Towards Component-Based Document-centric Software", Rich Lysakowski, Scientific Computing & Automation, October 1996.

12. "Programming Guide, Orbix 2", IONA technologies Ltd., pg 2, October 1996.

13. "Building Large-Scale CORBA-Based Systems", Rocky Stewart, Dave Rai, and Sanjay Dalal, Component Strategies magazine, pg 34, January 1999.

14. "Simulation of Beamline Alignment Operations", Mark G. Miller and Cynthia E. Annese, LDRD report, February 1999, UCRL-ID-133248.

15. "Countdown Status Messages Simulation", Cynthia E. Annese, LDRD report, October 1998, UCRL-ID-133242.

Work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-ENG-48.