ECE151 – Lecture 15

Chapter 9
Distributed Object-Based Systems
CORBA
CORBA

CORBA: Common Object Request Broker Architecture

Background:
Developed by the Object Management Group (OMG) in response to industrial demands for object based middleware
Currently in version #3
CORBA is a specification: different implementations of CORBA exist
Very much the work of a committee: there are over 800 members of the OMG and many of them have a say in what CORBA should look like
CORBA provides a simple distributed-object model, with specifications for many supporting services. In enterprises, it is legacy but is here to stay for a many years.
Overview of CORBA

The global architecture of CORBA.
Object Model

Object Request Broker (ORB): CORBA’s object broker that connects clients, objects, and services

Proxy/Skeleton: Precompiled code that takes care of (un)marshaling invocations and results

Dynamic Invocation/Skeleton Interface (DII/DSI): To allow clients to “construct” invocation requests at runtime instead of calling methods at a proxy, and having the server-side “reconstruct” those request into regular method invocations

---

**Diagram:**

- **Client machine**
  - Client application
    - Static IDL proxy
    - Dynamic Invocation Interface
    - ORB interface
  - Client ORB
  - Local OS

- **Server machine**
  - Object implementation
    - Object adapter
    - Skeleton
    - Dynamic Skeleton Interface
    - ORB interface
  - Server ORB
  - Local OS

---

Network
Object Model

**Object adapter:** Server-side code that handles incoming invocation requests.

**Interface repository:** Database containing interface definitions and which can be queried at runtime

**Implementation repository:** Database containing the implementation (code, and possibly also state) of objects. Effectively: a server that can launch object servers.

![Diagram of object model](image-url)
CORBA Object Model

CORBA has a “traditional” remote-object model in which an object residing at an object server is remote accessible through proxies

Observation: All CORBA specifications are given by means of interface descriptions, expressed in an IDL. CORBA follows an interface-based approach to objects: Not the objects, but interfaces are the really important entities An object may implement one or more interfaces

Interface descriptions can be stored in an interface repository, and looked up at runtime

Mappings from IDL to specific programming are part of the CORBA specification (languages include C, C++, Smalltalk, Cobol, Ada, and Java.)
## Corba Services

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection</td>
<td>Facilities for grouping objects into lists, queue, sets, etc.</td>
</tr>
<tr>
<td>Query</td>
<td>Facilities for querying collections of objects in a declarative manner</td>
</tr>
<tr>
<td>Concurrency</td>
<td>Facilities to allow concurrent access to shared objects</td>
</tr>
<tr>
<td>Transaction</td>
<td>Flat and nested transactions on method calls over multiple objects</td>
</tr>
<tr>
<td>Event</td>
<td>Facilities for asynchronous communication through events</td>
</tr>
<tr>
<td>Notification</td>
<td>Advanced facilities for event-based asynchronous communication</td>
</tr>
<tr>
<td>Externalization</td>
<td>Facilities for marshaling and unmarshaling of objects</td>
</tr>
<tr>
<td>Life cycle</td>
<td>Facilities for creation, deletion, copying, and moving of objects</td>
</tr>
<tr>
<td>Licensing</td>
<td>Facilities for attaching a license to an object</td>
</tr>
<tr>
<td>Naming</td>
<td>Facilities for systemwide name of objects</td>
</tr>
<tr>
<td>Property</td>
<td>Facilities for associating (attribute, value) pairs with objects</td>
</tr>
<tr>
<td>Trading</td>
<td>Facilities to publish and find the services on object has to offer</td>
</tr>
<tr>
<td>Persistence</td>
<td>Facilities for persistently storing objects</td>
</tr>
<tr>
<td>Relationship</td>
<td>Facilities for expressing relationships between objects</td>
</tr>
<tr>
<td>Security</td>
<td>Mechanisms for secure channels, authorization, and auditing</td>
</tr>
<tr>
<td>Time</td>
<td>Provides the current time within specified error margins</td>
</tr>
</tbody>
</table>

**Overview of CORBA services.**
## Object Invocation Models

<table>
<thead>
<tr>
<th>Request type</th>
<th>Failure semantics</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous</td>
<td>At-most-once</td>
<td>Caller blocks until a response is returned or an exception is raised</td>
</tr>
<tr>
<td>One-way</td>
<td>Best effort delivery</td>
<td>Caller continues immediately without waiting for any response from the server</td>
</tr>
<tr>
<td>Deferred synchronous</td>
<td>At-most-once</td>
<td>Caller continues immediately and can later block until response is delivered</td>
</tr>
</tbody>
</table>

Invocation models supported in CORBA.

ECE151 – Lecture 15
Event and Notification Services

The logical organization of suppliers and consumers of events, following the **push-style** model.
Event and Notification Services

The pull-style model for event delivery in CORBA.
Messaging

CORBA's callback model for asynchronous method

1. Call by the application
2. Request to server
3. Response from server
4. Call by the ORB
CORBA'S polling model for asynchronous method invocation.
# Interoperability

<table>
<thead>
<tr>
<th>Message type</th>
<th>Originator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Client</td>
<td>Contains an invocation request</td>
</tr>
<tr>
<td>Reply</td>
<td>Server</td>
<td>Contains the response to an invocation</td>
</tr>
<tr>
<td>LocateRequest</td>
<td>Client</td>
<td>Contains a request on the exact location of an object</td>
</tr>
<tr>
<td>LocateReply</td>
<td>Server</td>
<td>Contains location information on an object</td>
</tr>
<tr>
<td>CancelRequest</td>
<td>Client</td>
<td>Indicates client no longer expects a reply</td>
</tr>
<tr>
<td>CloseConnection</td>
<td>Both</td>
<td>Indication that connection will be closed</td>
</tr>
<tr>
<td>MessageError</td>
<td>Both</td>
<td>Contains information on an error</td>
</tr>
<tr>
<td>Fragment</td>
<td>Both</td>
<td>Part (fragment) of a larger message</td>
</tr>
</tbody>
</table>

GIOP message types.
Interceptors

Request-level: Allows you to modify invocation semantics (e.g., multicasting)
Message-level: Allows you to control message-passing between client and server (e.g., handle reliability and fragmentation)

Logical placement of interceptors in CORBA.
Naming

Important: In CORBA, it is essential to distinguish specification-level and implementation-level object references

Specification level: An object reference is considered to be the same as a proxy for the referenced object having an object reference means you can directly invoke methods; there is no separate client-to-object binding phase

Implementation level: When a client gets an object reference, the implementation ensures that, one way or the other, a proxy for the referenced object is placed in the client’s address space:

    ObjectReference objRef;
    objRef = bindTo(object O in server S at host H);

Object references in CORBA used to be highly implementation dependent: different implementations of CORBA could normally not exchange their references.
Interoperable Object References

**Observation:** Recognizing that object references are implementation dependent, we need a separate referencing mechanism to cross ORB boundaries.

**Solution:** Object references passed from one ORB to another are transformed by the bridge through which they pass (different transformation schemes can be implemented).

**Observation:** Passing an object reference \( \text{refA} \) from ORB A to ORB B circumventing the A-to-B bridge may be useless if ORB B doesn’t understand \( \text{refA} \).
Object References

To allow all kinds of *different* systems to communicate, we standardize the reference that is passed between bridges:
Object References

Indirect binding in CORBA.

IOR refers to implementation repository

1. First invocation or binding request
2. Activate/start object
3. Ack object is active
4. Redirect message
5. Actual invocation

Client

IOR

Object server

Implementation repository

ECE151 – Lecture 15
Naming Service

CORBA’s naming service allows servers to associate a name to an object reference, and have clients subsequently bind to that object by resolving its name.

**Observation:** In most CORBA implementations, object references denote servers at specific hosts; naming makes it easier to relocate objects.

**Observation:** In the naming graph all nodes are objects; there are no restrictions to binding names to objects => CORBA allows arbitrary naming graphs.

**Question:** How do you imagine cyclic name resolution stops?

**Observation:** There is no single root; an initial context node is returned through a special call to the ORB.

Also: the naming service can operate *across* different ORBs => **interoperable naming service**
Portable Object Adaptor

Mapping of CORBA object identifiers to servants.

a) The POA supports multiple servants.

b) The POA supports a single servant.
Portable Object Adaptor

My_servant *my_object; // Declare a reference to a C++ object
CORBA::Objectid_var oid; // Declare a CORBA identifier
my_object = new MyServant; // Create a new C++ object
oid = poa ->activate_object (my_object); // Register C++ object as CORBA OBJECT

Changing a C++ object into a CORBA object.
Fault Tolerance

**Essence:** Mask failures through replication of objects. Replicas form **object groups**.
Object groups are transparent to clients: they appear as “normal” objects.
This approach requires a separate type of object reference: **Interoperable Object Group Reference**:
Object Groups

IOGRs have the same structure as IORs, but different uses. In IORs an additional profile is used as an alternative; in IOGR, it denotes another replica.
An Example Architecture

An example architecture of a fault-tolerant CORBA system.
Security

Allow the client and object to be mostly unaware of all the security policies, except perhaps at binding time; the ORB does the rest. Specific policies are passed to the ORB as (local) objects and are invoked when necessary:

Examples: Type of message protection, lists of trusted parties.
The role of security interceptors in CORBA.