Distributed Objects

Object-Oriented Application Development

- Procedural (non-object oriented) development
  - Data: variables
  - Behavior: procedures, subroutines, functions
  - Languages: C, COBOL, Pascal
  - “Structured Programming” to avoid “spaghetti” code
Object-Oriented Application Development

- Programming with objects
  - An object represents both data and behavior
  - Data = attributes, behavior = methods
  - A collection of logically related data and behavior

Object-Oriented Application Development

- More than simply repackaging of procedural concepts
  - Encapsulation: inner workings of an object are not exposed (or "encapsulated")
  - Inheritance: an object automatically "inherits" attributes and behavior of its parent object; may redefine some attributes and behavior
  - Polymorphism: when invoked in the same way, different objects behave differently
Classes, Instances, Interfaces

- **Object Class vs. Object Instance**
  - **Class** is a blueprint, template for creating an object (program code on disk); e.g., Employee
  - **Instance** is a “live” object, with specific data (in memory); e.g., John Smith

- The concept of an Interface promotes encapsulation
  - Separates the implementation (inner workings) from its definition as seen by the outside world
  - The “contract” between the object and its users
  - “Pure abstract base class” in C++; directly supported by Java
  - Key concept for distributed objects
Non-Distributed Object-Oriented Development

- OO languages assume that all objects that make up an application exist in a single process space and are directly addressable.
- Monolithic applications living in a single process:
  - Word processor, graphics application, spreadsheet
- Language-based objects are completely unprepared for distribution:
  - Across OS processes
  - Across machines on the network

“Captive” Language Objects
Why Distributed Objects?

- Real world requires that applications be distributed
  - Performance and scalability reasons
  - Partitioning of application functionality
  - Reuse of application functionality

Why Distributed Objects?

- Application developers want to use OO concepts in developing distributed applications
  - Reuse through encapsulation and inheritance
  - Peer-to-peer messaging interfaces are too low-level
- Distributed object environments make objects first-class citizens
  - Not bound to single OS process by the language environment
Objects in Distributed Environments

- OO language environments rely on a concept of an object reference
- Object attributes (data) and methods (behavior) are accessed with the help of the reference
- Somewhere underneath it is a memory pointer
- Memory pointer makes no sense in distributed environments
- Different processes have their own memory address space

Communications across networks is based on messages
Distributed Object environments provide the glue to extend the concept of an object reference across process spaces and networks
Distributed Object Bus: A Pipe

- The Object Request Broker bus is just a connectivity pipe
- It will allow objects to communicate
- A lot more is needed for building distributed applications: distributed object services

Distributed Object Bus: A Pipe

- Typical distributed services needed by objects:
  - Naming - How to locate objects and connect to them
  - Transactions
  - Security
  - Lifecycle - When to create and delete objects
  - Persistence - How to store object attributes
  - Etc.
Distributed Objects and Services

Distributed Objects and Runtime Infrastructure

- A distributed object exists in a runtime environment
- Runtime infrastructure concerns include:
  - Who starts and stops the program that contains objects?
  - Who manages connections between objects?
  - Who manages concurrent access by multiple clients?
  - Who provides scalability and availability?
  - Many others...
Distributed Objects and Runtime Infrastructure

Application Runtime

Object

Naming

Transactions

Security

Other

Object Bus

Naming

Transactions

Security

Other

Application Runtime

Distributed Objects: CORBA-Style

- Common Object Request Broker Architecture
- Developed by OMG (Object Management Group)
- Consortium of 800+ companies (not Microsoft)
- Committee creates specs
- Microsoft has its own distributed object infrastructure: DCOM (COM+)
- Sun (JavaSoft) is both a CORBA supporter and detractor
  - Java defined many competing standards to CORBA
  - Love-hate relationship...
**CORBA IDL**

- IDL: Interface Definition Language
- Language-neutral specification of an object’s interface
- Only declares the interface (“contract”),
- CORBA IDL takes roots in DCE RPC IDL
- RPC IDL is used to describe the functional interface of a procedure being accessed remotely via RPC
- All services and objects that are part of CORBA standard are described in IDL
- IDL is the first step to allow objects written in different languages to interoperate

**CORBA IDL Example**

```idl
interface Simple
{
    string to_lower(in string val);
    void to_upper(inout string val);
};
```
CORBA IDL Language Bindings

OMG Object Management Architecture
**CORBA 2.0 ORB**

**CORBA ORB: Client Side**
- Client IDL stubs
  - Representation of the server object on the client
  - Marshals (packs into messages) invocations and calls ORB runtime
- Dynamic Invocation Interface (DII)
  - For discovering and invoking objects at runtime
- Interface Repository API
  - Obtain descriptions of registered interfaces
- ORB Interface
  - Utility methods
**CORBA ORB: Server Side**
- Server IDL Stubs (skeletons)
  - Unmarshals the invocations and calls the object
- Dynamic Skeleton Interface (DSI)
  - Server counterpart of DII
- Object Adapter
  - Basic runtime environment
  - Assigns object references and registers objects with implementation repository
  - BOA and POA
- Implementation Repository
  - Information about object classes supported by the server, instantiated objects and their IDs.

**CORBA IIOP**
- Internet Inter-ORB Protocol
- General Inter-ORB Protocol over TCP/IP
- Before CORBA 2.0 and IIOP, ORBs from different vendors couldn’t talk to each other
- IIOP defines the standard protocol for communication between ORBs
- Some implementations also use IIOP internally
Evolution of CORBA

RMI