CORBA 3
and
CORBA For
Embedded Systems

Dr. S. Ron Oliver
U. S. Representative for Top Graph'X
sroliver@topgraphx.com www.topgraphx.com

- Your Instructor

- Attendees
  - Your job function
  - Your computing experience
  - Your expectations

Introduction

- Objective of the Tutorial
  - Understand overall CORBA concepts
  - Learn the basic mechanisms
  - Learn the application of CORBA extensions/refinements for Embedded Systems

Introduction

- Overview of Course Content:
  - The OMG
  - CORBA, the ORB
  - IDL, the Ada95 mapping
  - Object References, from the viewpoint of the POA
  - The POA in detail
  - Exceptions, callbacks
  - Interface and Implementation Repositories
  - Naming Service
  - Event Service

Introduction

- Overview of Course Content, continued:
  - GIOP
  - The OrbAda environment
  - CORBA For Embedded Systems
  - minimumCORBA
  - CORBA Messaging Specification
  - Real Time CORBA

The OMG and Interoperability

- What is the OMG ?
- Why a distributed architecture ?
- Interoperability
- CORBAservices
- CORBAfacilities

What is the OMG?

- The biggest computer industry consortium
- Created in April 1989
- A small team
- Mission: Create and promote object oriented standards for the integration of applications running on existing computing technologies.

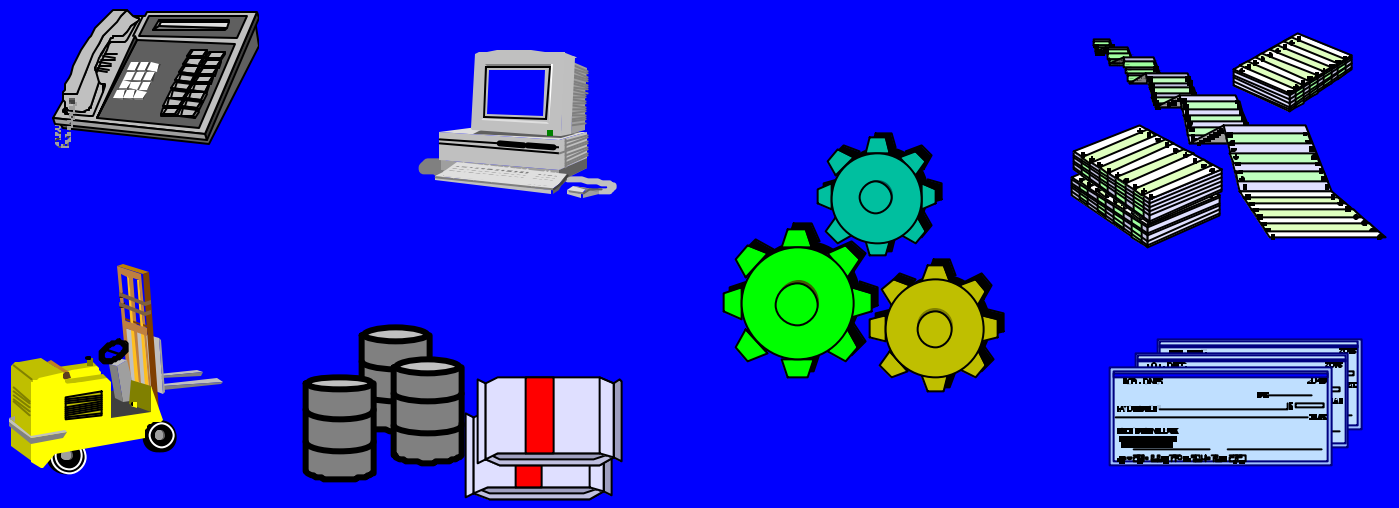
What is the OMG ?

- The suite of OMG specifications:
  - Why a distributed architecture ?
  - Portability and applicability of OMG specifications

Note : Some of the transparencies in this course were composed from documents provided by the OMG to its members.

Why a Distributed Architecture ?

- Distribute software for Functionally separate entities !



Why Interoperability

A consensus on interoperability is therefore necessary.

CORBAservices

- Naming Service and Trader Service
  - Passing around object references
- Event Service and Notification Service
  - Notification when events happen
- Object Transaction Service
  - Transaction processing applications
- Security Service
  - Controlling access to corporate assets
- Messaging Service
  - Critical for Real Time systems
- Other Services
  - Nine have been identified

CORBAfacilities

- Horizontal Facilities
  - General applicability
  - User interface common facilities
  - Information management common facilities
  - System management common facilities
  - Task management common facilities
- Vertical Facilities
  - Domain specific
  - Examples:
    - Business Objects Work Flow Management Facility
    - CORBAmed's Person Identifier Service

What is CORBA ?

- Characteristics of CORBA
- Characteristics of an object
- Interface Definition Language (IDL)
- OMG/ISO IDL
- The Basis of CORBA interoperability

What is CORBA ?

- The Object Request Broker (ORB)
- ORB interfaces
- ORB components
- The client side
- The implementation side
- Inter ORB Interoperability
- CORBA - Interoperability

Characteristics of CORBA

- Independence :
  - Platform/vendor
  - Operating System
  - Protocol (hardware/software)
  - Location
  - Programming Languages
- Strong Typing and Run Time Dispatching
- Object Orientation :
  - Encapsulation
  - Polymorphism
  - Inheritance
  - Instantiation

Characteristics of an Object

- An Object :
  - A combination of functionality and state
  - An abstraction of a real world object
  - Has a well defined interface
  - An “object” reference or address
  - Adheres to the fundamental principles of OO :
    - Encapsulation Inheritance
    - Polymorphism Instantiation
  - Specification of a Class of objects
  - Object semantics is critical

IDL - Isolation

IDL isolates the interface from the implementation

Object Request Broker

IDL - Specification

Object Request Broker **IDL**

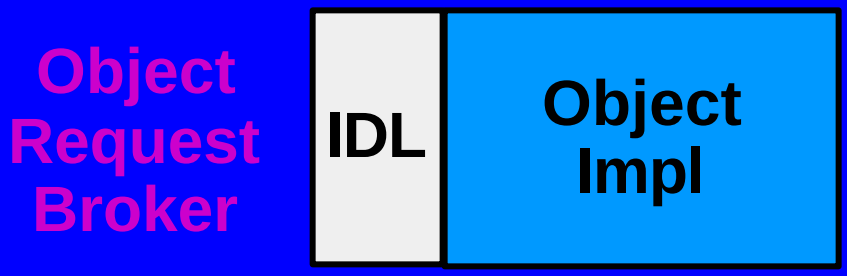
Define the interface of an object using IDL

Operations, Parameters, Types, Exceptions

This defines a language independent API for the object

IDL –Languages

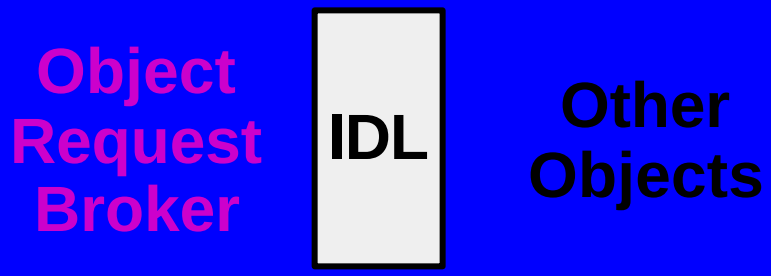
Separation of interface and implementation



Objects may be developed in different programming languages :
-- C, C++, Java, Smalltalk, Ada, COBOL, Visual Basic, etc . . .

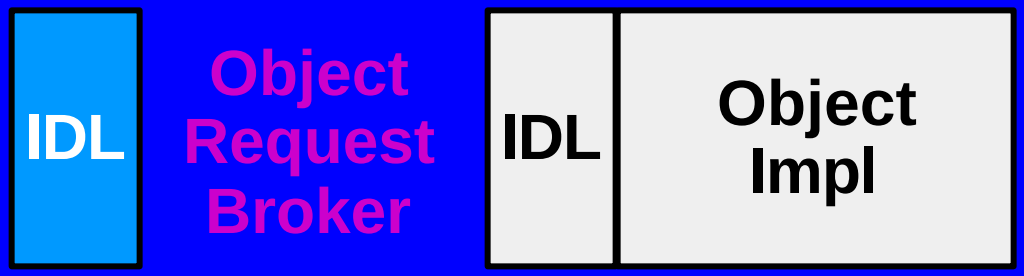
IDL –Objects

Separation of interface and implementation



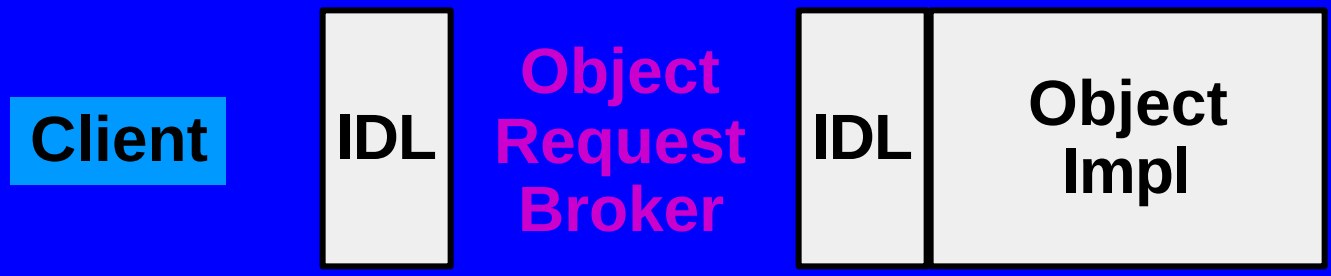
Object implementations may be existing applications, encapsulated; may be generated by utilities; may be acquired from a vendor.

IDL – Uniqueness of View



The same IDL defines the API as seen by the client.

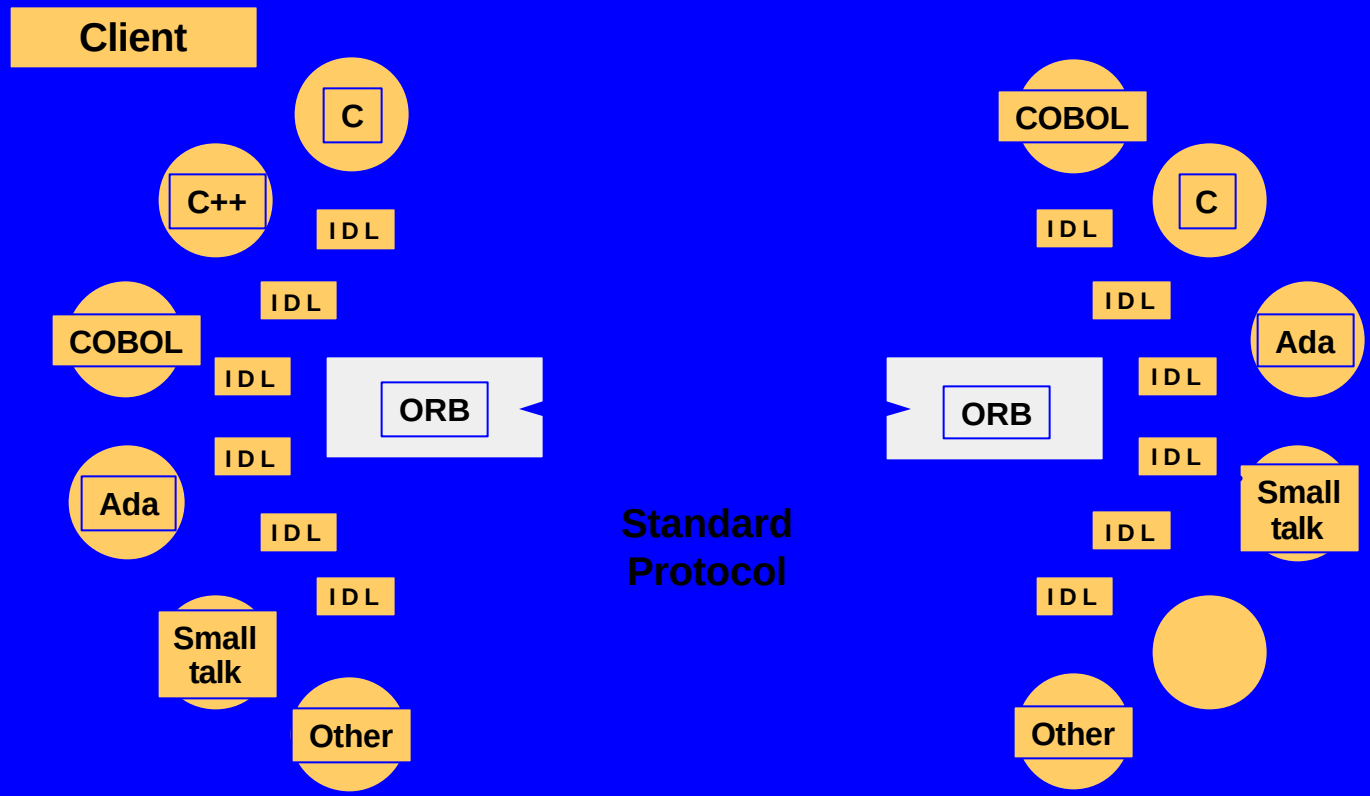
IDL – Clients



Just as in the case of object implementations, clients may be developed from scratch, generated by utilities, or purchased.

IDL - Role

- The link between ORBs
- Coexistence of different languages and systems



OMG/ISO IDL

- Separates the interface from the implementation:
  - A language for specifying public interfaces.
  - Strongly typed and supports multiple inheritance.
  - Mappings are available for a number of languages.
  - *Not a programming language.*
- Makes interoperability possible

The Basics of CORBA Interoperability

- Interoperability via inter-ORB communication
- An OMG Standard Protocol
- General Inter – ORB Protocol (GIOP)
  - Common Data Representation (CDR)
  - Message formats
  - Transport requirements
- Internet Inter – ORB Protocol (IIOP)
  - GIOP over TCP / IP

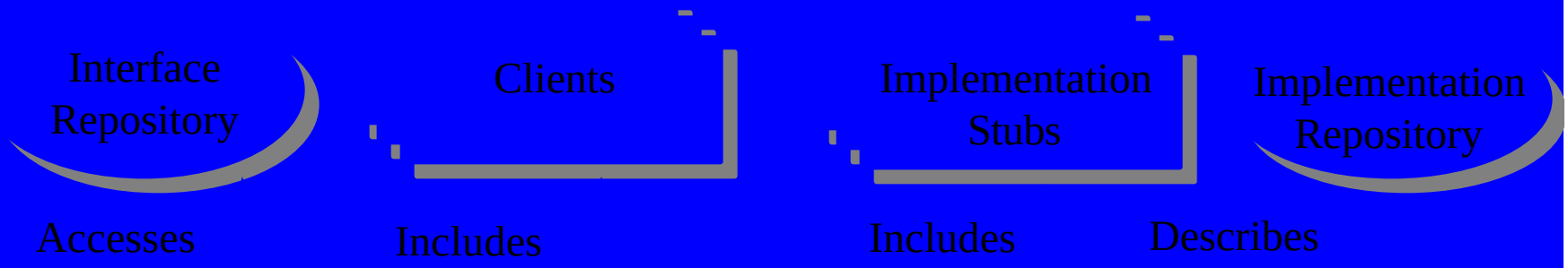
The Object Request Broker

- Components of an ORB
- The Portable Object Adapter
- Support for Dynamic Invocations
- Interface Repository
- Client API and Server Stubs

ORB Interfaces

IDL Interface Definitions

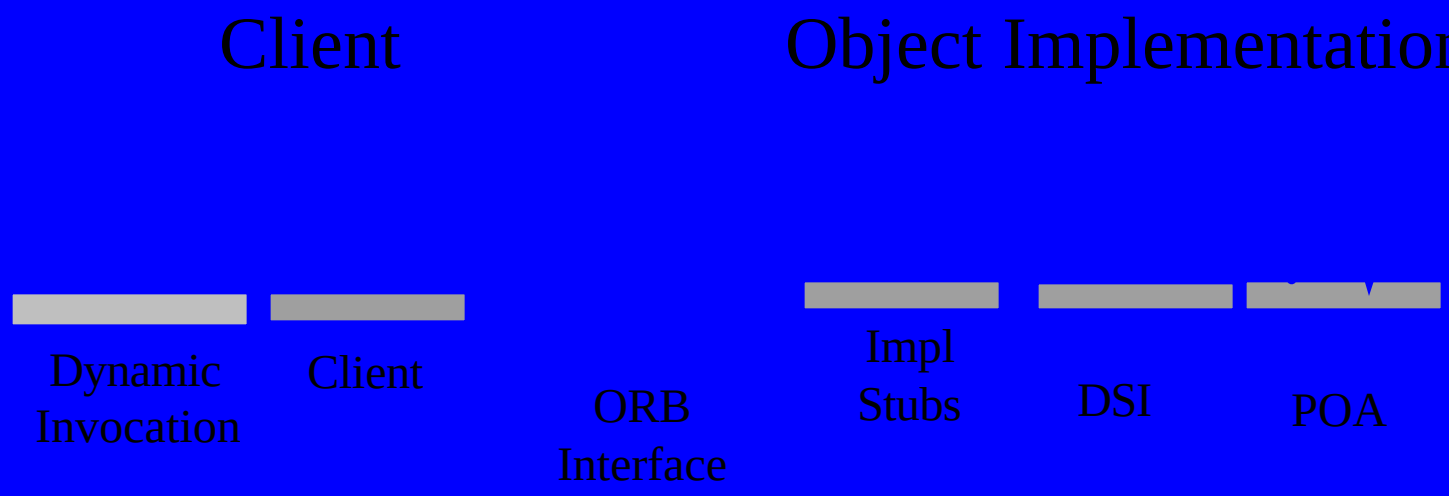
Implementation Installation



Client

Object Implementation

Components of the ORB



One interface

One interface per object adaptor

One interface per object operation

Proprietary interface

Normal call interface

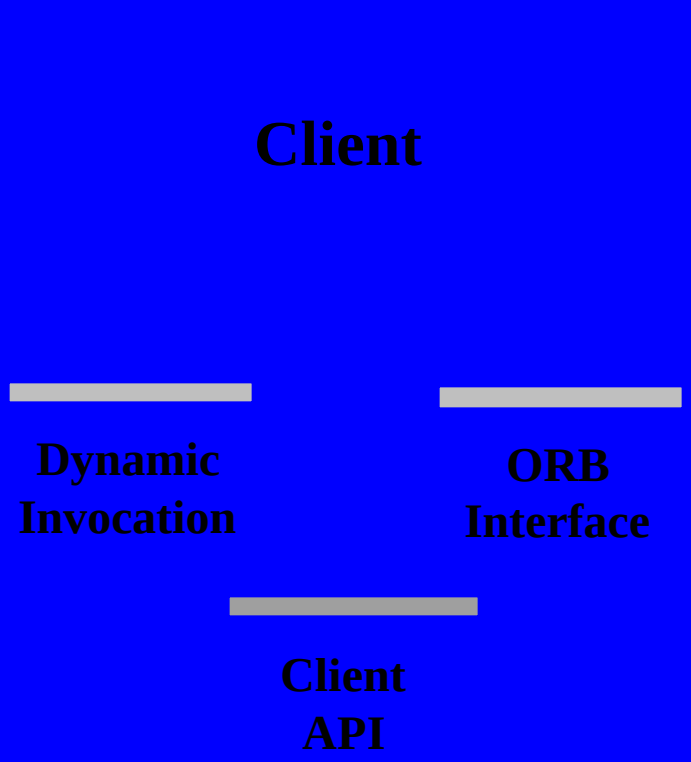
Up call interface

DSI: Dynamic Skeleton Interface

POA: Portable Object Adapter

The Client Side

A client issues requests using object references.



A client issues requests via the API using Static Methods (Static Invocation Interface – SII) or the Dynamic Invocation Interface (DII)

A client obtains general services from the ORB :

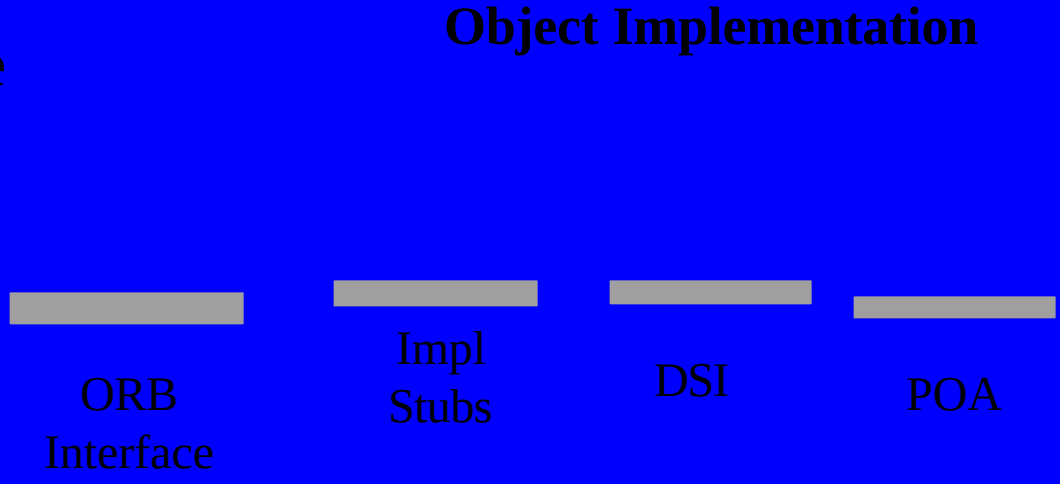
- Interface Repository
- Context Management
- Request Management

The Implementation Side

Implementations receive requests via their stubs (without knowledge of the client)

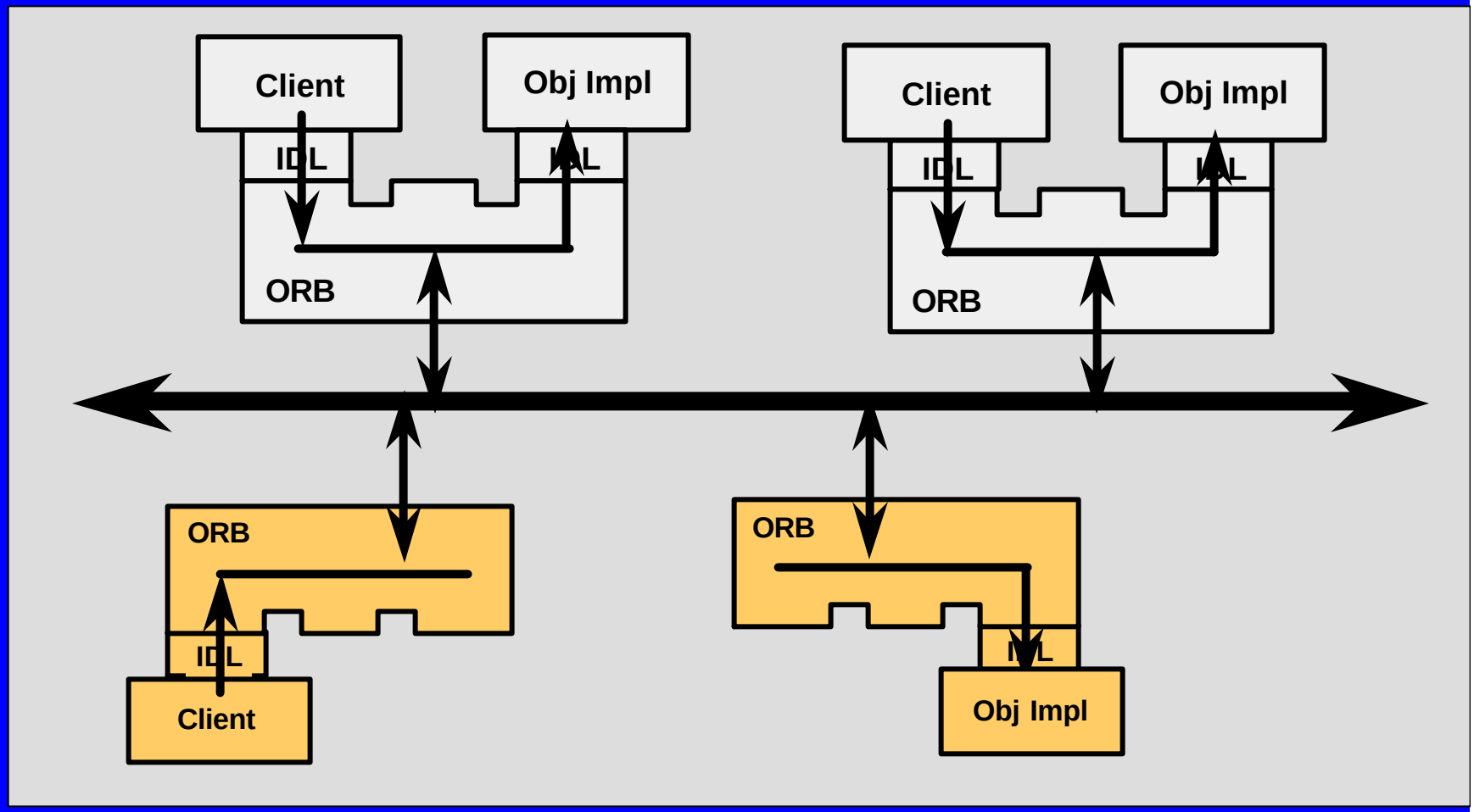
The POA (Portable Object Adapter) offers a large number of models of implementation with one portable format.

The POA supports both SII and DII in the same way.



Inter ORB Interoperability

- The general case



CORBA - Interoperability

CORBA interoperability consists of:

- An all encompassing architecture for ORB-ORB communication;
- One API for adding bridges;
- One multi-transport message format (General Inter-ORB Protocol or GIOP);
- One API for routers using ESIOPs -- (Environment-Specific Inter-ORB Protocols)

The Universal, Turn-Key Interoperability :

- IIOP – that is, GIOP over TCP/IP – is required for conformity, either in an internal form or via a bridge,
- Specialized protocols are optional and well supported by the specification.

The Elements of IDL

- **IDL - Presentation**
- **IDL - Files**
- **Modules**
- **Interfaces**
- **Inheritance**
- **Operations**
- **Attributes**
- **Data Types**
  - Basic
  - Constructed
  - Template
- **Pragmas**

IDL - Presentation

- A declarative language used to describe object interfaces.
- Independent of the implementation language of the object or client which uses it.
- An interface written in IDL provides all information about operation parameters.
- Purely declarative, with no information about algorithmic structures or local variables.
- A source file containing IDL type definitions will normally have extension “.idl”.
- The file orb.idl is provided with all ORB implementations.

IDL – Presentation

- Lexical rules based on the declarative part of C++ (with new keywords)
- C++-like Syntax for the declaration of constants, types and operations
- Key Word Convention :
  - You may use IDL Key words without conflict
  - Prefix : '_'
  - Example : _long

IDL - Files

- May contain many interfaces
- May include other IDL files
- May contain pre-processing directives
  - (as in C or C++)
    - #include
    - #ifdef
    - #define

    etc.
- Are compiled by the IDL compiler

  - Ex : idl2ada [-options] myfile.idl

Modules

- Name Space
  - Note : an identifier may only be defined once in the same Name Space
- Contain IDL declarations
- May contain other modules
- Key Word : module
  - Example :

```
module  Finance {
… <<one or more Interfaces>>
};
```

Interfaces

- Definition of Object Characteristics
  - Types
  - Attributes
  - Operations (methods)
  - The Unit of inheritance
- May inherit other interfaces
- An object implements one or more interfaces.
- Example :

```
interface Bank {
… <<one or more type, operation, attribute, and / or
  exception declarations>>
};
```

Inheritance

- Applies to interfaces
- Example

```
interface vehicle {
        long op1 (in long arg1)
};
interface passenger_car:vehicle {
        void op2 (in short arg2)
};
```

Operations

- Methods of the object
- Parameter passing mode is specified
- Return Type is required (but may be void)
- Synchronous unless it is oneway
- Key Words :

  oneway

  in, out, inout

  raises

Attributes

- Identical in form to an IDL variable, except :
  - IDL compiler automatically generates the operations set and get for an attribute.
- Unless it is declared *readonly* -
  - Only the get operation is available.
- A good way to build up state.
  - But not the only one !

Basic Types

- Integer:
  - short, long, long long
  - signed & unsigned
- Floating point:
  - float, double,
    long double, fixed
- char, wchar, boolean, octet

Constructed Types

- Structs
- Unions
- Enums
- Arrays
- Any

Template Types

- Sequence
- String
- Wstring
- Fixed
  - Scale
  - Precision

Example

- A grocery store that sells produce by the kilogram, for U. S. currency
- Current stock includes potatoes, carrots, bananas, and oranges
- An order form is provided
- A Customer may
  - Inquire the price of a kilogram of an item
  - Order a list of groceries to be delivered
  - Close the store

Store IDL

```
interface Store
{
#pragma OrbAda_Directive "Threading" "False"
#pragma OrbAda_Directive "Root_Needed" "True"
#pragma OrbAda_Directive "Activation" "Unshared_Server"

    typedef float Kilos ;
    typedef float Dollars ;

    enum Goods { Potatoes, Carrots, Bananas, Oranges};
    struct Order_Type
    {
        Kilos Potatoes ;
        Kilos Carrots ;
        Kilos Bananas ;
        Kilos Oranges ;
    };

    Dollars Price (in Goods Kilo) ;

    void Deliver (in Order_Type Order, out Dollars
                  Invoice);

    oneway void Close () ;
};
```

What Store IDL Generates When Run Through idl2ada

- Package Store
  - Complete
- Package Store–skeleton
  - Complete
- Package Store–helper
  - Complete
- Package Store–impl
  - Template
  - Developer must fill in bodies of procedures (price, deliver, close)

Store Implementation

```
function Price
   ( Self : access Object;
     Kilo : in Goods)
   return Dollars is
begin
   return Self.Price (Kilo) ;
end Price ;

procedure Deliver
   ( Self : access Object;
     Order : in Order_Type;
     Invoice : out Dollars) is
begin
Invoice := Dollars(Order.Potatoes) * Self.Price (Potatoes)
   + Dollars (Order.Carrots) * Self.Price(Carrots)
   + Dollars (Order.Bananas) * Self.Price (Bananas)
   + Dollars (Order.Oranges) * Self.Price (Oranges) ;
end Deliver ;

procedure Close
   ( Self : access Object) is
begin
   Corba.Orb.Stop ;
end Close ;
```

Pragmas

- IDL Compiler directives
  - ID : sets the repository ID of the IDL entity for the Interface Repository
  - Version : sets the version of the IDL entity for the Interface Repository.
  - Others : Implementation Dependent.

IDL /Ada95 Mapping

- Base Packages
- IDL Containers
- Methods
- Complex Types
- The idl2ada Compiler

Base Packages

- Basic Types : defined in package Corba with their operations
- Corba.Object: package which contains the reference base class
- Corba.Orb
- Corba.PortableServer

IDL Containers

- IDL File
  - If there are declarations not contained in an interface or Module, we generate an Ada Package with the same file name plus the suffix '_IDL_FILE'
- Module
  - An Ada Package
- Interface
  - Client side :
    - Package interface (contains the reference class) + the Helper
  - Server side :
    - Child Package Impl (contains the servant class)
    - Others if necessary (implementation dependent)
      - Ex : '.Skeleton' for OrbRiver/Ada

Methods

- Operations
  - To each operation there corresponds a reference primitive and a servant primitive in their respective packages
- Attributes
  - Get Primitive (function)
  - Set Primitive (procedure) if not ‘readonly’

Complex Types

- Arrays
  - An Ada array where ‘first is equal to 0
- Sequences
  - Bounded
    - Instance of the generic Corba.Sequence.Bounded
  - Non-bounded
    - Instance of the generic Corba.Sequence.Unbounded

Complex Types

- Structures
  - Ada record
- Unions
  - Variant record
- Exceptions
  - Ada exception + an Ada class derived from Corba.Idl_Exception_Members
  - Get_Members Primitive

Mapping of basic types

IDL	CORBA	ADA
short	CORBA::Short	Corba.short
long	CORBA::Long	Corba.long
unsigned short	CORBA::UShort	Corba.Unsigned_short
unsigned long	CORBA::ULong	Corba.Unsigned_long
float	CORBA::Float	Corba.float
double	CORBA::Double	Corba.double
char	CORBA::Char	Corba.char
boolean	CORBA::Boolean	Corba.Boolean
octet	CORBA::Octet	Corba.Octet
Enum	Enum	Ada enum
wchar	CORBA::wchar	Corba.wchar

Client Implementation

- Object References
- Interoperable Object Reference (IOR)
- Client Initialization
- Obtaining an object reference

Object References

Knowing how to access an object

In order to invoke an object method it is first necessary to get a reference to the object.

Idea : To be able to search for the object in a directory

Problem : To get a reference to the directory !!

Chicken and egg problem !!

Interoperable Object Reference (IOR)

- Portable binary form of an Object reference
  - It is the form exchanged with the CDR protocol.
- Represented in the form of a character string
  - Corba.Orb.Object_To_String
  - Corba.Orb.String_To_Object
- Structure of the IOR
  - Repository ID of the object interface
  - List of operation profiles of the object
    - IIOP Profile : hostname, TCP port, object key
    - Shared Memory Profile, multicast (specifically OrbRiver)

Client Initialization

Initialization of the ORB

Corba.Orb.Orb_Init (Args, OrbName)

- The procedure processes the command line by calling
  - Ada.Command_Line
- Args permit you to pass values
  - taken first – may force some values
- OrbName : null string or name of the ORB

Client Example

- A Customer Client
  - Initializes the ORB
  - Accesses the known Store Service / Object
  - Retrieves the price of carrots
  - Orders 1 kilogram of each product
  - Closes the Store
  - Terminates Connection with the ORB

Customer Code

```
with Ada.Text_Io ;
with Store ;
with Corba.Orb ;
with Corba.Object ;
procedure Customer is
   Args        : Corba.Orb.Arg_List ;
   Orb_Name    : Corba.Orb.Orbid ;
   A_Store     : Store.Ref ;
   Price       : Store.Dollars ;
begin
   -- First connect the ORB
   Corba.Orb.Orb_Init (Args, Orb_Name) ;

   -- Get the Store as an initial service
   Corba.Orb.Resolve_Initial_References
      (Store.Tgx_Service_Name, A_Store);
```

Customer Code cont.

```
   -- Apply required methods
   Price := Store.Price (A_Store, Store.Carrots) ;
   Ada.Text_Io.Put_Line ("1 kg of carrots costs $" &
                         Store.Dollars'image (Price)) ;

   Store.Deliver ( A_Store, Store.Order_Type'(
                   others => 1.0), Price) ;
   Ada.Text_Io.Put_Line (
            "A bag with 1 kg of all products costs $" &
                         Store.Dollars'image (Price)) ;

   Store.Close (A_Store) ;

   -- End the CORBA processing
   Corba.Orb.Stop ;
end Customer ;
```

Execution Model of the POA

- Client
  - Context of requests for objects and other entities
- Server
  - Context of an object implementation; typically in a process, a task is at any given instant in the role of server or client.
- Object
  - CORBA view; simple for the client, but not for the server.

Execution Model of the POA

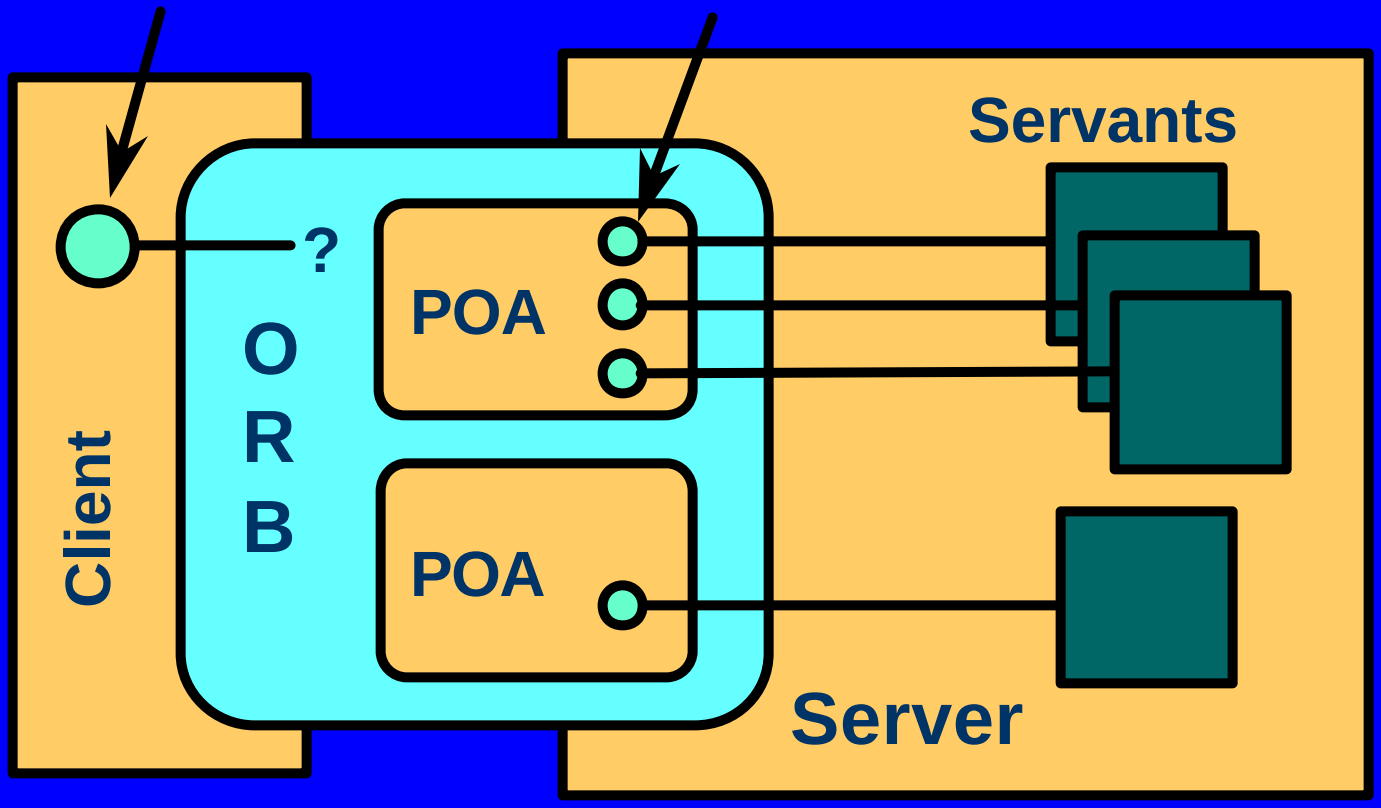
- Servant
  - An implementation in a given programming language. An object can be associated with one or many Servants, and this relationship may change over time.
- Object ID
  - Value used by the POA or the Servant to identify a particular CORBA object. Not visible to the clients.
- Object Reference
  - The CORBA IOR, encapsulates an “Object ID” and a “POA ID”
- POA
  - One of the server entities, equipped with its own name space and policies. The POAs form a hierarchical structure.

Execution Model of the POA

- Policy
  - Controls how a POA interacts with Servants and other entities
- POA Manager
  - An object written by the Vendor to be used by the ORB to manage POAs and their Servants, to queue up or reject requests. Provides request flow control.
- Servant Manager
  - An object written by the Developer to be used in the POA to manage Servants.
- Adapter Activator
  - An object written by the Developer, called by the ORB.
  - The “Adapter Activator” creates a child POA .

Execution Model of the POA

- The POA identifies the Servant which must execute the requests sent by the Client.



The Portable Object Adaptor

- POA
  - Possesses a local reference and a unique name.
  - 4 States (Holding, Active, Discarding, Inactive).
  - The behavior is configurable by the policies.
  - Manages a table of active objects.
    - The Active Object Map (AOM)
  - Belongs to a hierarchy of POAs stemming from « RootPOA ».

What does a POA provide ?

- Portability of object implementations from one ORB to another
- Object instances may have Persistent Identities
- Transparent Activation of objects
- Multiple Instances implemented by a unique Servant
- Transient objects with minimal implementation
- Reasonably good control of behavior and persistence
- Multiple policies for fundamental objects

Creation of a new POA

- There is a “ *Root* POA”
  - Managed by the ORB
  - Uses default policies.
- A new POA is created :
  - To choose different policies
  - To use its own “ adapter activator”
  - Or its own “ servant manager”
  - To control the treatment of requests in an independent way
- Support for Persistent Objects
  - Recreate the POA (and its ancestors) which initially created the object reference and supported the Servant in its previous “life”.

Using the “ Root POA”

```
-- Standard CORBA method for locating root POA
--    and POA manager.
-- You’ll use these three lines in every CORBA server
--    you write:
-- Only way to get Root POA Reference
      Corba.Orb.Resolve_Initial_References
      (Corba.To_Unbounded_String ("RootPOA"), Root_Poa) ;
      Manager :=
      Corba.PortableServer.Poa.Get_The_POAManager
      (Root_Poa) ;
-- Instantiate servant (here, an Ada operation):
      Root_Ptr :=       Perfs.Impl.Object ;
```

Using the " Root POA" cont.

-- Activate the poa_manager. You'll use this line

-- in every server too:

```
Corba.PortableServer.PoaManager.Activate (Manager) ;
```

-- Implicit activation of servant, returning

-- object reference:

```
Corba.Object.Ref (Root_Ptr) :=
Corba.PortableServer.Poa.Servant_To_Reference

( Self      => Root_Poa,        P_Servant =>
Corba.PortableServer.Servant (Root_Ptr));
```

-- We're done with POA stuff. Now we can register

-- the object reference With the naming service,

-- and invoke the object.

POA Policies

- ThreadPolicy
  - ORB_CTRL_MODEL
  - SINGLE_THREAD_MODEL
- LifespanPolicy
  - TRANSIENT
  - PERSISTENT
- IdUniquenessPolicy
  - UNIQUE_ID
  - MULTIPLE_ID

POA Policies

- IdAssignmentPolicy
  - SYSTEM_ID
  - USER_ID
- ImplicitActivationPolicy
  - IMPLICIT_ACTIVATION
  - NO_IMPLICIT_ACTIVATION

POA Policies

- ServantRetentionPolicy
  - RETAIN
  - NON_RETAIN
- RequestProcessingPolicy
  - USE_ACTIVE_OBJECT_MAP_ONLY
  - USE_DEFAULT_SERVANT
  - USE_SERVANT_MANAGER

Activation of an Object

- Call *activate_object* or *activate_object_with_id:*
- `ObjectId activate_object ( in Servant p_servant)`
  - Activates the servant, generates and returns the OID. The object is registered in the AOM
  - Only makes sense if Servant Retention Policy is RETAIN
- `void activate_object_with_id ( in ObjectId oid, in Servant p_servant)`
  - Activates the designated servant by the OID provided. The object is registered in the AOM

RETAIN and USE_ACTIVE_OBJECT_MAP_ONLY

- The servant activates all objects by a call to *activate_object* or *activate_object_with_id*
  - If IMPLICIT_Activation and System_ID are also set, it may activate objects via *servant_to_reference* or *servant_to_id*
- The POA searches the table of active objects to find the Servant for a request.

RETAIN and USE_SERVANT_MANAGER

- RETAIN permits use of *activate_object* or *activate_object_with_id*
  - If IMPLICIT_Activation and System_ID are also set, it may activate objects via *servant_to_reference* or *servant_to_id*
- If there is no servant registered in the AOM, the POA invokes *incarnate* on the ServantActivator registered as ServantManager

RETAIN and USE_DEFAULT_SERVANT

- There is a default Servant for all requests for all unknown objects in the AOM.
- By using RETAIN, the application is able to initialize servants in the AOM by calling *activate_object* or *activate_object_with_id*
  - If IMPLICIT_Activation and System_ID are also set, it may activate objects via *servant_to_reference* or *servant_to_id*
- The POA first searches the AOM to find a servant, if it is not found, it uses the default servant.

NON-RETAIN and USE_SERVANT_MANAGER

- There is no AOM
- The POA calls *preinvoke* operation of the ServantLocator registered as ServantManager to find a servant
- After request is processed POA calls *postinvoke* operation of the Servant Locator (same parameters) even if there is an exception

NON-RETAIN and USE_DEFAULT_SERVANT

- A single servant defined for all objects
- There is no AOM
- There is a default servant attached to the POA, and it is used for all requests.
- All objects have the same interface, and are stateless
- Simple, efficient, safe

What is the Purpose of a Servant ?

- NOT one reference for one implementation
- For example :
  - To deal with several millions of operations per hour you are might simultaneously use several implementations of the same object.
  - When one must deal with 4 million commands for different objects, a single servant implementation or a small number might be sufficient.

The Servant

- You can change the server side model without affecting the client.
- A servant is an implementation which, when it executes, provides functionality for one or more object references.
- Policies allow you to determine what messages pass between servants and Objects.
- The “Object Ids” (OIDs) permit you to identify the Servants attached to the POA
- The POA is able to activate/deactivate/manage Servants.

The Object ID

- Different from the IOR:
  - The Client never knows an OID
- In reality, it is included in IORs of Persistent Objects.
- Can be assigned by the developer.
- Can identify Persistent Storage (database key, file name, etc)
- Also used by the POA .

Servant Example

- The Store Servant:
  - Connects to the ORB
  - Activates the Root POA
  - Initializes the Store Service
    - Makes the Store Implementation active
    - Initializes Store Variables, etc.
  - Sets up the Store Object Reference
  - Waits for requests
  - Disconnects from the ORB

Store Servant Code

```
with Store.Impl ;
with Corba.Orb ;
with Corba.PortableServer.Poa ;
with Corba.PortableServer.PoaManager ;
procedure Store_Example is
   Root     : Store.Impl.Object_Ptr ;
   Root_Ref : Store.Ref ;
   Root_Poa : Corba.PortableServer.Poa.Ref ;
   Manager  : Corba.PortableServer.PoaManager.Ref ;
   Args     : Corba.Orb.Arg_List ;
   Orb_Name : Corba.Orb.Orbid ;
begin

   -- First connect the ORB
   Corba.Orb.Orb_Init (Args, Orb_Name) ;

   -- Get the root POA and activate it
   Corba.Orb.Resolve_Initial_References
      (Corba.To_Unbounded_String ("RootPOA"), Root_Poa) ;
   Manager := Corba.PortableServer.Poa.Get_The_POAManager (Root_Poa);
   Corba.PortableServer.PoaManager.Activate (Manager) ;

   -- Initialize the service
   Root := new Store.Impl.Object ;
```

Store Servant Code

```
-- Here, please fill in the root object
Root.Price:= ( Store.Potatoes => 1.0,
               Store.Carrots  => 1.5,
               Store.Bananas  => 2.0,
               Store.Oranges  => 2.5) ;

-- Associate a reference with the object
Corba.PortableServer.Poa.Servant_To_Reference
   ( Self      => Root_Poa,
     P_Servant => Corba.PortableServer.Servant (Root),
     Result    => Root_Ref);

-- Make it an Initial service
Corba.Orb.Add_Initial_Reference
   ( Identifier => Store.Tgx_Service_Name,
     Object     => Root_Ref) ;

-- Start method processing
Corba.Orb.Run ;

-- End the CORBA processing
Corba.Orb.Stop ;
```

Exceptions

- CORBA provides standard exceptions which are system exceptions
  - Example in package Corba :

    Object_Not_Exist : exception;

- The developer can define his own exceptions
  - Exception BarcodeNotFound {POS::Barcode item};

Exception Parameters

- A Corba exception can have parameters
  - Example in package Corba :

```
type Object_Not_Exist_Members is new
   System_Exception_Members with null record;
```

  - Procedure get_members (. . .)
- Furnishes state of execution achieved :
  - The field can have the following values :
    - Completed_Yes :
      Occurred after the request was processed
    - Completed_No :
      Occurred before processing of the request
    - Completed_Maybe
      Information is indeterminate

Callbacks

- Definition
  - A method implemented for a client and called by a Server
- Use of *oneway methods*:
  - The client does not block when it calls the method (optional)
  - The Server does not wait for the client to be available when it calls the callback (very important).

Callbacks - Example

- Use Symmetric interface

```
module TeleSurveillance {
    interface Client {
        oneway void wakeme_up (in string event);
    };
    interface Guard {
        void start_surveillance (in Client some_where);
    };
};
```

Using Callbacks

- Instantiating Surveillance

```
MyCellPhone : TeleSurveillance.Client;
```

- When leaving the office

```
TeleSurveillance.Guard.start_surveillance( MyCellPhone );
```

- When the event occurs

```
some_where.wakeme_up( whichever_event )
```

Dynamic Interfaces

The Dynamic Invocation Interface (DII) allows clients to access objects without previous knowledge of them or their interfaces

The Dynamic Skeleton Interface (DSI) is the object-side construct corresponding to the DII

Interface Repository

- Role :
  - Dynamic access for a client to an interface in order to construct a request.
  - Dynamic control of type signatures of requests.
  - Walk the inheritance tree.
  - Inter-ORB interoperability.

Access to the IR

- In general, 2 methods :
  - Standard OMG interface
  - Specific interface used by the ORB (utilities)
- Comment from OMG
  - The second method is the most common …
- OrbRiver uses the standard interface !!!

Identification of an IR

- It is possible to use many IR
  - An ORB can know about more than one IR, and
  - A single IR can likewise be referenced by many ORBs
- Advice: use at least 2 IR
  - One for the Production System
  - Another for Development

Implementation Repository

- Permits you to manage the activation of objects
  - Manual activation on the Server (human intervention or activation scripts launched by the system)
  - Automatic activation on the Server by the ORB according to *Activation Policies*.

Activation Policy : Persistent

- Activation is controlled by the User

- Key Points :
  - The service is presumed to always be available
  - Not activated when you call a method.

Activation Policy : Shared

- A single instance of the Server for all client applications.
- Server automatically activated when needed.

- Key Points :
  - Shared data.
  - Appropriate for multi-tasking.

Activation Policy : Unshared

- One server instance per client application

- Key Points :
  - Separate data
  - Best for single thread

Naming Service

- Used to propagate object references down the pipeline.
- Provides the CorbaName service.

CORBA URLs

- Defined by the Interoperable Naming Service
  - Not strictly a part of CORBA 3
- Two formats:
  - corbaloc
  - corbaname
- Resolved by string_to_object

CORBALOC

- Gets an Object Reference for a Standard Service at a Remote Location
- Format:

  corbaloc://1.0@MyInternetVendor.com:2809/NamingService

- Some fields may be defaulted:

  corbaloc://MyInternetVendor.com/NamingService

CORBANAME

- Invokes the Naming Service at a Remote Location

- Format:

corbaname://MyInternetVendor.com/Pub/Menswear/SaleCatalog.obj

- Same Defaults as corbaloc

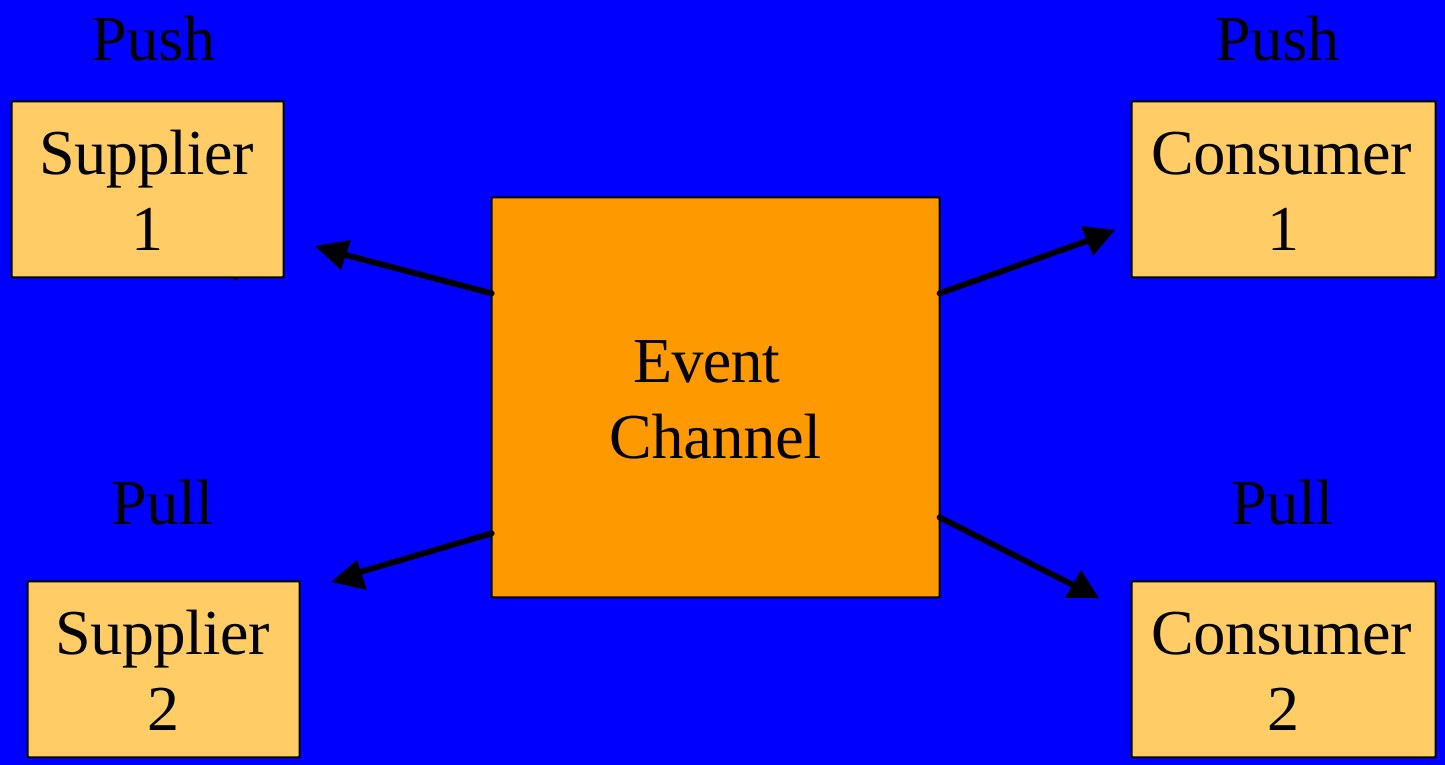
Event Service

- Asynchronous Mechanism
- Decouples communication from Objects
- Defines 2 roles for Objects
  - Supplier
  - Consumer
- Defines 2 models of communication
  - Push
  - Pull

The Event Channel

- An Event Channel is a service which decouples communication between suppliers and consumers.
- One or many Suppliers; one or many Consumers
- It is itself a Consumer and Supplier of Events
- Events include a piece of data, type ***any***
  - Typed Event Channels may be used, also
- There may be multiple Event Channel objects

The Event Channel



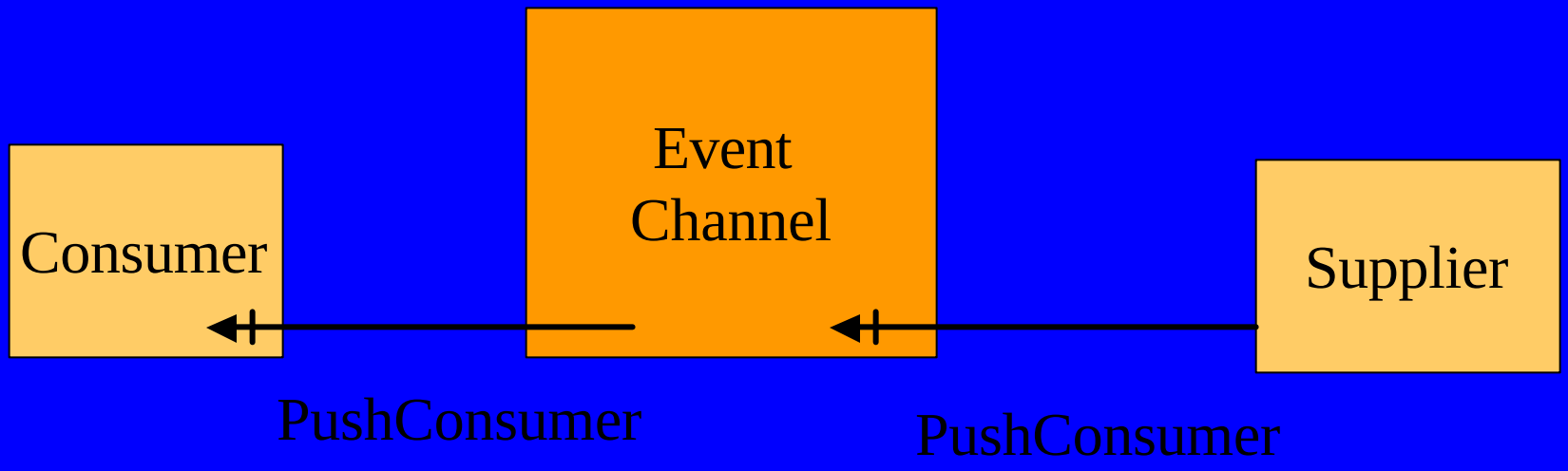
Event Channel Example

- One type ***any*** Channel
- 2 Suppliers, 2 Consumers
- Push, Pull Model
- 4 interfaces

Push Supplier, Push Consumer

- Event Channel is Push Consumer
  - Supplier is Client, Channel is Server
  - Supplier Pushes event to Channel
- Event Channel is Push Supplier
  - Channel is Client, Consumer is Server
  - Channel Pushes event to Consumer
- Event Channel is Pull Supplier
  - Channel is Server
  - Pushed events queued until Pulled

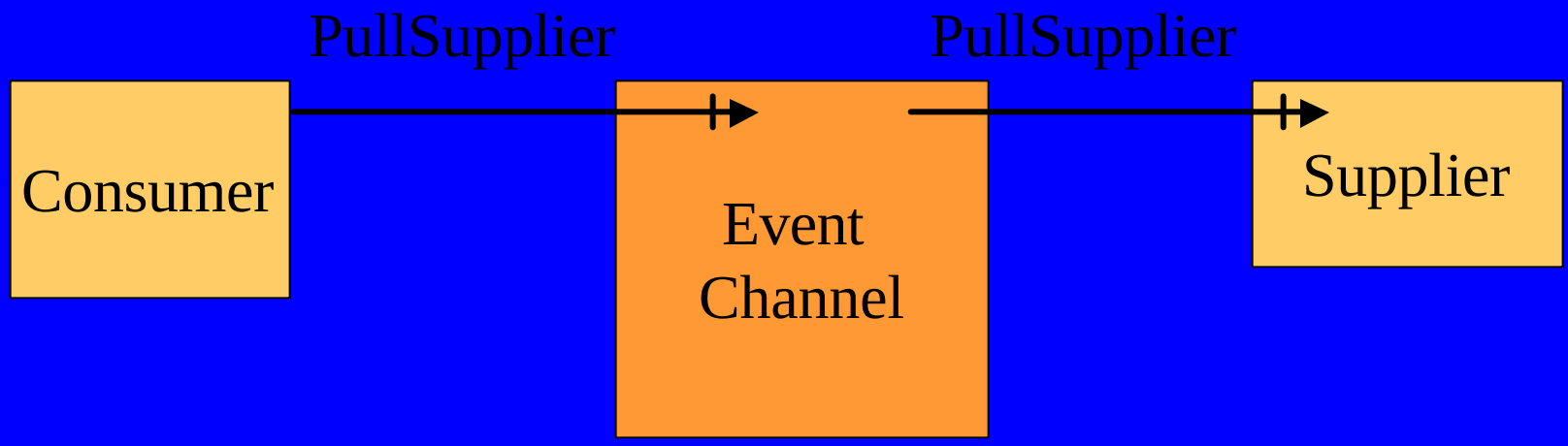
Push + Event Channel



Pull Consumer, Pull Supplier

- Event Channel is Pull Supplier
  - Channel is Server, Pull consumer is Client
  - Consumer Pulls events from Channel
  - Pull Blocks, if no event
  - Can use **try_pull** to avoid blocking
- Event Channel is Pull Consumer
  - Attempts to Pull events from Pull Suppliers
  - Channel is Client, Pull Supplier is Server
  - Channel is also Push Supplier for Pulled Events

Pull + Event Channel



Notification Service

- Inherits Event Service Interfaces
- Supports 3 data types
  - any
  - Struct
  - Sequence
- Provides Quality of Service control
  - Event Reliability
  - Connection Reliability
  - Priority
  - Time management
  - Others

Components of GIOP

- Eight GIOP Messages

SENT BY CLIENT:

- Request
- LocateRequest
- CancelRequest

SENT BY OBJECT:

- Reply
- LocateReply
- CloseConnection

SENT BY EITHER:

- MessageError
- Fragment

Components of GIOP

- CDR: Common Data Representation
  - Standard wire representations and typecodes for every OMG IDL datatype
  - Includes dynamic-length types such as structs and strings, and the CORBA Object Reference
- Transport Requirements:
  - Reliable
  - Connection-oriented
  - Connection initiation follows TCP/IP model
  - View as byte stream
  - Notification of connection loss

The OrbRiver Environment

- The installation directory - Tree

bin	utilities
doc	html files
etc	configuration
examples	
idl	specifications of the service provided
omg_doc	OMG reference documents

OrbRiver/Ada – the daemon

- A multi-function process :
  - ***multi threaded ORB***
  - ***Implementation Repository :*** stores information about implementations in a persistent form and makes dynamic execution of services possible.
  - ***Interface Repository :*** stores information about interfaces in a persistent form and makes their use via dynamic invocation possible.
  - ***Naming Service :*** implements the complete OMG specification. Its objects are persistent.
  - ***Debug utility :*** permits the analysis of requests and their responses.
  - ***Licensing Service :*** Manages license keys for using the product.

OrbRiver/Ada - Utilities

- Add_license
- Idl2ada
- Ior_dump
- Impl_editor
- Ir_editor
- Machine_id
- Name_editor
- OrbRiver/Ada
- Stop_orb
- Debugger
- All in one (in Java)

CORBA For Embedded Systems

- Embedded Systems
- minimumCORBA
- CORBA Messaging Specification
- Real Time CORBA
- OrbRiver-Critical

Embedded Systems

- One or more processors without “standard” keyboard and CRT interfaces
- Usually support several sensor and / or device controller interfaces
  - Aircraft
  - Automobile engines
  - Microwave ovens, etc.
- Usually 2 or more processors interacting via some communication “channel”
- Usually incorporate features of Concurrent and / or Real Time Systems
- Often are mission critical

Concurrent and Real Time Systems

- Concurrent: two or more processes on one or more processors, cooperating to accomplish some task(s)
  - Synchronization Mechanisms
  - Parallel operation when possible
- Real Time: one or more parts of a task must be completed within a specific time delay or by a specific time of day
  - Time management operations required
  - Resource management required
  - Priority management often necessary

Concurrent Processes

- When using two or more processors
  - Distributed system
  - Communication “channel”
- Often run, or appear to run, in parallel
- Use some protocol(s) to interact and avoid conflict over shared resources
  - Rendezvous
  - Semaphores

Communication “Channel”

- Layers of hardware and software components cooperating to effect processor-to-processor communication
- Protocols: rules and procedures via which hardware/software components manage exchange of Data
- Physical Layer: establishes bound on information throughput
- Transport Layer: imposes orderly shared use of link and physical layer
  - Connection mechanisms - Naturally adds overhead
  - Error control and reliability

Resources Shared by Concurrent Processes

- Data (in Memory), devices, files, library routines, Communication “channel”
- Correct adherence to protocols required for correctness of cooperating Concurrent Processes (C.P.)
  - Client / Server Model
- C.P. are generally *much* more complicated than other processes
- Very little about C.P. is taught in most C.S. programs

Real Time Processes

- Process correctness depends on meeting Real Time requirements
  - A “correct” computation that completes late is **incorrect**
- Adherence to Real Time requirements must often be done within small tolerances
  - Updating a digital clock display
  - Controlling airplane thrust or maneuvers
- Real Time Processes are often Concurrent Processes, and vice versa

minimumCORBA

- Rationale
- Profile Philosophy
- IDL and ORB Interface
- Dynamic Features
- Interoperability

minimumCORBA Rationale

- Embedded Systems tend to:
  - Have restrictive code size requirements
    - Limited memory and or storage resources
  - Require predictable behavior
    - To ensure Real Time requirements may be met
- minimumCORBA is a strict subset of Corba
  - Compatibility
- Dynamic features are removed
- Retain maximum compatibility with full CORBA

minimumCORBA Profile

- A single profile only, targeted at a broad range of limited resource systems
- Vendors may offer further link-time flexibility

minimumCORBA IDL and ORB Interface

- IDL is retained in full
  - Maximum compatibility
- A number of omissions are made from the ORB Interface
  - Operations that support the DII
  - Operations needed only for certain styles of CORBA applications, not for basic ORB operation
    - Workpending, perform_work, shutdown
  - Operations that support the Interface Repository

Dynamic Features Omitted from minimumCORBA

- Dynamic Invocation Interface
- Dynamic Skeleton Interface
- Interface Repository (Majority of it)
  - Part of the dynamically typed programming work

minimumCORBA Interoperability

- The same interoperability conformance criteria as Full CORBA
- DCE Interoperability is omitted
- COM / CORBA Internetworking is omitted
- Interceptors are omitted (depends on DII and DSI)
- Language Mappings
  - Must support at least one, unspecified language
  - Some C++ and Java Mapping features are omitted

CORBA Messaging Specification

- Terminology
- Quality of Service Model
- Messaging Programming Model
- Message Routing Interoperability

CORBA Messaging Terminology

- Synchronous
  - Client blocks until reply received
- Deferred Synchronous
  - Client Blocks only until request delivered to target
  - Reply may be returned later
- Time–Independent
  - Deferred Synchronous but reply may be received in different computational context from that of invocation

CORBA Messaging Terminology (cont.)

- Asynchronous
  - Client does not block at all
- Quality of Service (QoS) features
  - Delivery quality
  - Queue Management
  - Message Priority

CORBA Messaging QoS

- QoS Framework
- Messaging QoS
- Propagation of Messaging QoS

Message QoS Framework

- All QoS Settings derived from CORBA :: Policy
- Client-side Policies – 3 contexts
  - ORB – level
  - Thread – level
  - Object – level
- Server-side Policies are associated with the POA
- There are various exceptions and interfaces available for accessing and managing QoS policies

Messaging Quality of Service Policies

- Rebind Support
- Synchronization Scope
  - Transport, ORB, Server, Target
- Request and Reply Priority
- Request and Reply Timeout
  - Start, End, Elapsed
- Routing
- Queue Ordering

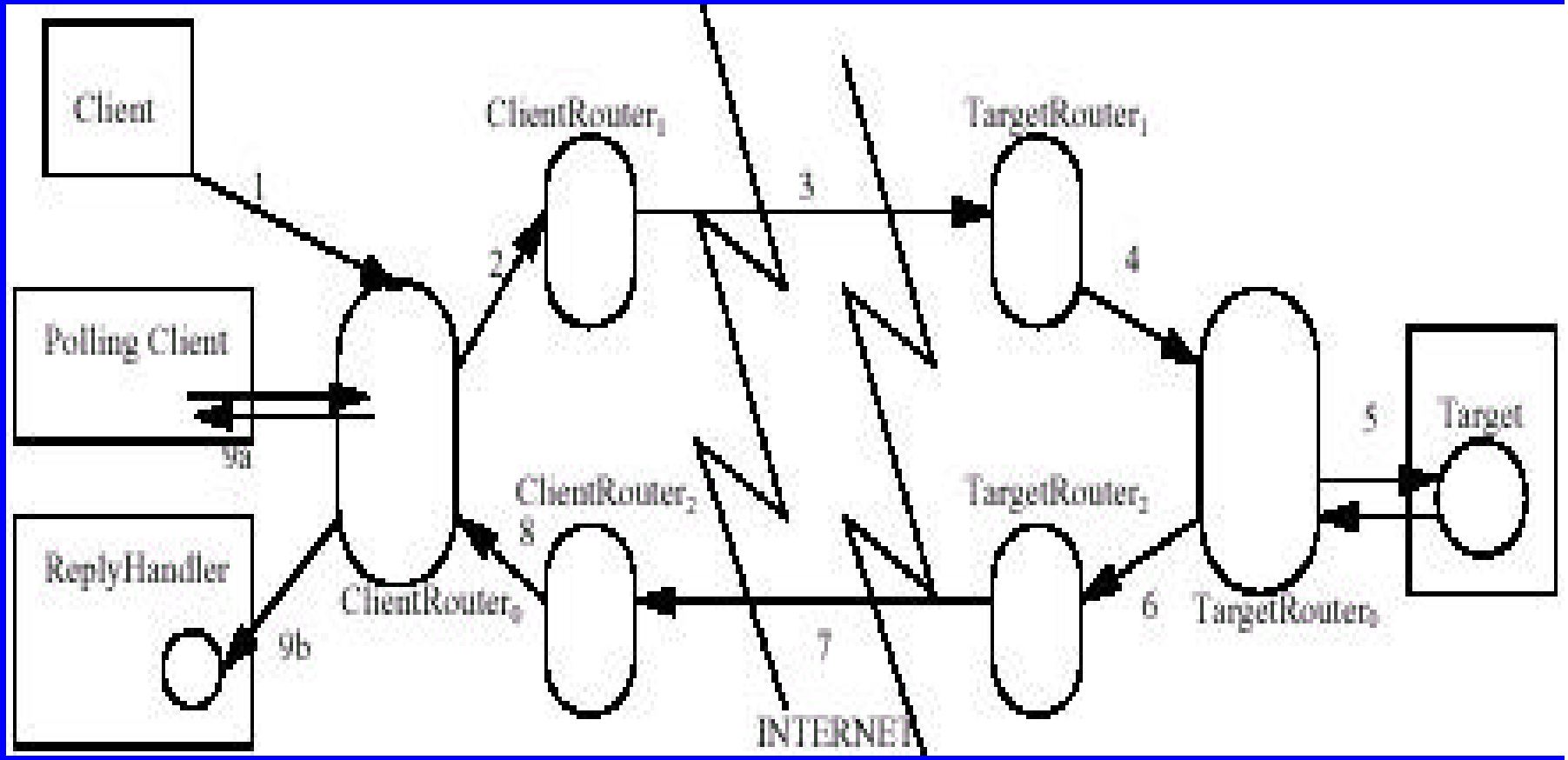
CORBA Messaging Programming Model

- Traditional Invocation Model
  - Synchronous
  - Oneway
- Asynchronous Method Invocation
  - Deferred Synchronous
    - Polling
      - Queryable Poller
  - Asynchronous
    - Callback
      - Reply Handler

CORBA Message Routing

- Routers
  - Accept requests from clients
- Interact with other Routers via a Communication "channel"
  - (e.g., the Internet)
- Make Requests of Target Objects
- Similarly for Replies
- Routing Policies
  - None
  - Forward
  - Store and Forward (Persistent)

Routing Interoperability Overview



Real Time CORBA

- Architecture
- Extensions
- Scheduling Service

Real Time CORBA Architecture

- Depends on Messaging Specification
  - Policy Framework (QoS)
  - Managing Timeouts
  - Asynchronous Method Invocation
- Deterministic Behavior
  - ORB – level
  - Application Level
- “activity” – analysis / design concept
- End-to-End Predictability
- Management of Resources

End-to-End Predictability

- Timeliness Control Needed :
  - Respect thread priorities to resolve resource contention
  - Bound the duration of priority inversions
  - Bound operation invocation latency
- System Components
  - Scheduling mechanisms of the OS
  - Real Time ORB
  - Communication “channel”
  - The application
- Tools

Real Time CORBA Extensions

- Real Time ORB and POA
- Priority Management
- Threads and Thread Pools
- Connection Management

Real Time ORB and POA

- RT ORB – Extension of standard ORB
  - RT and Non-RT applications may coexist
- May specify priority range on initialization
- RT POA – Extension of Standard POA
  - Provides operations for managing object-level priorities
  - Understands Real Time Policies, as well as others

Native Thread Priorities

- OS sufficient to support RT CORBA will have a mechanism for representing thread priorities
- Discrete range of values
- Direction for higher priority
- Base vs. Active native thread priority management
- Priority inheritance protocol
  - Must be used by RT ORB
  - Specific protocol is implementation-dependent

CORBA Priority

- Short range 0 to 32767
- Higher value is higher priority
- Vendor must provide some procedures:
  - To-Native
  - To-CORBA
  - May not raise exceptions
  - Apply to clients, threads, POAs

RT CORBA Priority Models

- Client Propagated
- Server Declared
- Set as a Policy of the POA
- May be overridden on a per-object basis
- User-defined Priority Transforms may be used to implement different Priority Models

Threads and Thread Pools

- Thread Pools may be created
  - Stack size, number of static threads, and number of dynamic threads
  - Default priority
  - Allow or disallow request buffering
  - Maximum number of buffered requests
  - Maximum request buffer size
- Thread Pools may have Lanes
  - Lanes have separately set priorities,
    - Numbers of static and dynamic threads
  - May allow or disallow borrowing threads
    - Borrow from a lane of lower priority

Connection Management

- Implicit Binding
  - Caused by the ORB “when needed”
  - May entail undesirable overhead or delay
- Explicit Binding
  - Use validate_connection to force binding to occur say, during start-up
- Priority Banded Connections
  - May be used to set up connections with different ranges of priorities
- Private Connection may be requested
- Transport Protocol may be selected

Conclusion

- General principles of CORBA
- Writing IDL modules
- Creating Clients and Servers
- The OrbRiver environment
- CORBA for Embedded Systems

References

- ***CORBA 3 - Fundamentals and Programming***
  Written and Edited by Jon Siegel

- ***ESSENTIAL CORBA, The: System Integration Using Distributed Objects*** .
  Thomas J. Mowbray, PhD and Ron Zahavi.
  Published by John Wiley and Object Management Group. 1995

Useful WEB Sites

- **OMG**
  - www.omg.org *The Bible !*
- **Specific References available there:**
  - ***minimumCORBA*** ***orbos/98-08-04***
  - ***CORBA Messaging*** ***orbos/98-05-05***
  - ***Real-Time CORBA*** ***Joint Revised Submission orbos/99-02-12***