

Distributed Object Systems

– 6 –

Corba/Activation/POA

Piet van Oostrum

Sept. 30, 2008

Corba

Today:

- ▶ Interaction with the ORB
- ▶ Object references
- ▶ Activation
- ▶ Object Adapters
- ▶ Implementation Repository

Later:

- ▶ Naming Service
- ▶ Dynamic Invocation
- ▶ Interface Repository
- ▶ RMI-IIOP

Piet van Oostrum

DOS-6

DOS-6

1

Interaction with ORB

- ▶ Every Corba program must initialize its ORB:

```
import org.omg.CORBA.*;
import java.util.Properties;
...
public static void main(String args[]) {
    Properties props = new Properties();
    ...
    ORB orb = ORB.init(args, props);
    ...
}
```

- ▶ args can contain specific options for the ORB
- ▶ You can put more options in the props

Piet van Oostrum

DOS-6

DOS-6

2

ORB init

- ▶ The call to ORB.init is in general system/language dependent
- ▶ Most systems pass the command line args to ORB.init
- ▶ ORB.init picks up the options it understands and removes these
 - ▶ e.g. -ORBiiop.port=2343
- ▶ The properties can be used to give additional information, e.g. to use another Corba implementation (JacORB):

```
prop.setProperty("org.omg.CORBA.ORBClass",
                "org.jacorb.orb.ORB");
prop.setProperty("org.omg.CORBA.ORBSingletonClass",
                "org.jacorb.orb.ORBSingleton");
props.setProperty("iiop.port", "2343");
```

- ▶ Properties can usually also be set by other means (e.g. orb.properties in the JVM directory)

Piet van Oostrum

DOS-6

DOS-6

3

Object references

How do you reference a Corba object?

- ▶ Inside a program you have the reference to the stub. This cannot be used outside of the program
- ▶ On the server side the ORB has some kind of reference that contains information about how to find the object. This generally cannot be used with other ORBs.
- ▶ Interoperable Object reference (IOR) is a ORB-independent way of specifying an object. It encodes the host (IP address), port number and an index in the ORB.
- ▶ Conversion to/from a string by `object_to_string` and `string_to_object`.
- ▶ In the IIOP communication a similar (binary) value is used.

Piet van Oostrum

DOS-6

DOS-6

4

How to communicate object references?

- ▶ Use IOR
 - ▶ Put it in a file and let the client read it
 - ▶ Put it on a web site where clients can find it
 - ▶ Email it
 - ▶ Print it and give it on the clients command line
- ▶ Use a naming service
 - ▶ Naming service is also an object
 - ▶ How do we find the object reference of the naming service?
 - ▶ Put it in a file, web site etc.
 - ▶ Call `resolve_initial_references` in the ORB.
 - ▶ Use an Object URL

Piet van Oostrum

DOS-6

DOS-6

5

Finding the Naming Service

- ▶ `resolve_initial_references`

```
org.omg.CORBA.Object objRef =
    orb.resolve_initial_references("NameService");
NamingContext ncRef = NamingContextHelper.narrow(objRef);
```

The ORB must know where the naming service is located (e.g. through configuration file)

- ▶ There are a few other Initial services that can be obtained in this way (e.g. Interface Repository)
- ▶ `list_initial_services()` gives a list of these.

Initial services Pseudo IDL (PIDL)

```
module CORBA {
//list_initial_services
typedef string ObjectId;
typedef sequence <ObjectId> ObjectIdList;
ObjectIdList list_initial_services ();
//resolve_initial_references
exception InvalidName {};
Object resolve_initial_references
    (in ObjectId identifier)
    raises (InvalidName);
}
```

reserved ObjectIds are RootPOA, POACurrent, InterfaceRepository, NameService, TradingService, SecurityCurrent, TransactionCurrent, DynAnyFactory, ORBPolicyManager, PolicyCurrent, NotificationService and TypedNotificationService.

Object URLs (1)

- ▶ Purpose: identify object by an URL-like string
- ▶ `corbaloc` can be used to find objects with
 - ▶ hostname
 - ▶ port where the ORB is listening
 - ▶ object key with which the object is registered in the ORB (can be any octet sequence, is not interpreted by the ORB)
 - ▶ There is no standard way in a server to connect an object key to an object!!
- ▶ Form is: `corbaloc://xloo.cs.uu.nl:900/NameService` or `corbaloc:iiop://1.2@xloo.cs.uu.nl:900/NameService`
 - ▶ `iiop` is the protocol to use
 - ▶ `1.2@` denotes the IIOP version
- ▶ `corbaloc:rir:/NameService` means: get the object reference from `resolve_initial_references`

Object URLs (2)

- ▶ `corbaname` uses the Naming Service
- ▶ `corbaname:xloo.cs.uu.nl:900#GDOS.examples/Echo.object`
- ▶ This looks up the Naming Service at `xloo.cs.uu.nl` port 900 for the object with name `GDOS.examples/Echo.object`.
- ▶ The Naming Service to be used by default is the `/NameService` object at the host/port (like in `corbaloc`).
- ▶ If no host/port is given the `corbaloc:rir:/NameService` is used
- ▶ Note: Corba 2.3 uses `//` before the hostname, 2.4 omits these.
- ▶ Object URLs can be given as argument to `string_to_object`.

Example

```
org.omg.CORBA.Object objRef =
    string_to_object("corbaloc:rir:/NameService");
NamingContext ncRef = NamingContextHelper.narrow(objRef);
```

```
org.omg.CORBA.Object objRef = string_to_object(
    "corbaloc:iiop:xloo.cs.uu.nl:2809/NameService");
NamingContext ncRef = NamingContextHelper.narrow(objRef);
```

Activation

- ▶ A distributed object system may support millions of objects
- ▶ Not all of them have to be in use at all times
- ▶ Objects that are not 'active' do not need to use system resources
- ▶ E.g. each item in a database could be an object but not all of them need to be loaded in memory
- ▶ **Transient object** = object that does not survive stopping and starting of the server
- ▶ **Persistent object** = object that survives stopping and starting of the server
- ▶ **Active object** = running somewhere
- ▶ **Passive object** = persistent object that is not (yet) active
- ▶ **Activation** = making a passive object active

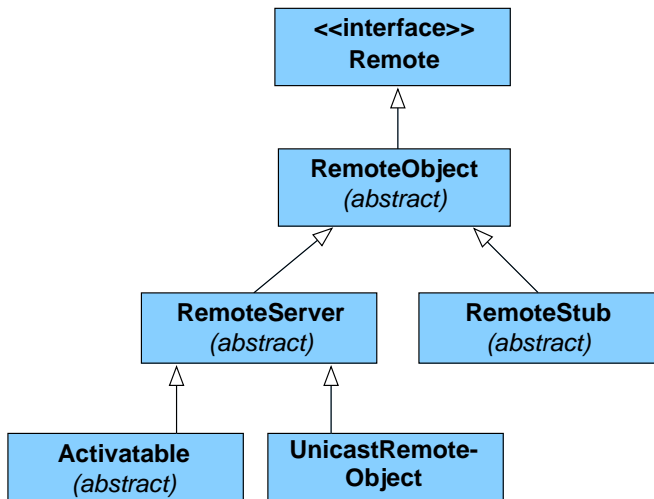
Activation of objects

- ▶ An object can belong to a running server where it is to be activated
- ▶ Or the server (process) can also be stopped
- ▶ A remote reference must contain some information that allows the object to be activated
 - ▶ This could contain the location of the server that the object has to run in
 - ▶ Or else a special server that can start other servers
 - ▶ Also additional information and identification of the object (e.g. primary key)

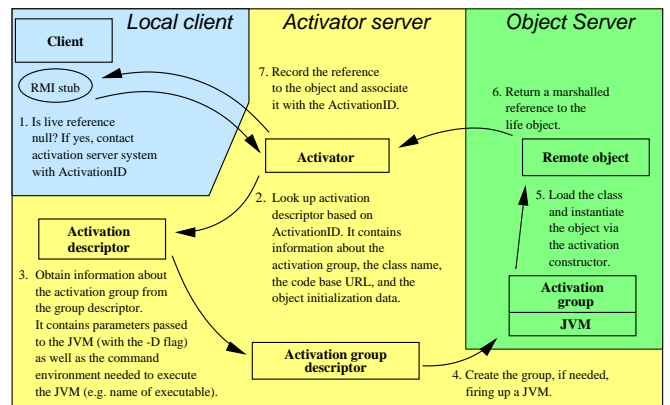
Activation in RMI

- ▶ Special server `rmiid`
- ▶ is also RMI name server
- ▶ can start the server (JVM) to run the object
- ▶ object can be registered by another server
- ▶ client stub contains a special reference ('faulting remote reference') with two parts
 - ▶ a persistent handle (ActivationID)
 - ▶ a transient remote reference 'live reference' (= null if passive or unknown)
- ▶ Involves:
 - ▶ Activator (activation server)
 - ▶ Activation group
 - ▶ Activation descriptor

Activation in RMI (2)



Activation in RMI (3)



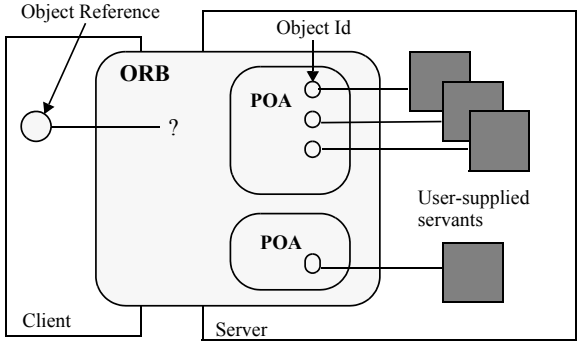
Activation Protocol actors

- ▶ Activator:
 - ▶ database of mapping ActivationID → running info
 - ▶ manager of JVM's
- ▶ Activation Group:
 - ▶ object (one per JVM) that activates passive objects
- ▶ Activation Descriptor:
 - ▶ Group ID → JVM
 - ▶ class name
 - ▶ URL for class code
 - ▶ object initialization data (marshalled)
- ▶ `rmiid` registers activation descriptors

Corba Object Adapters

- ▶ Object Adapters are intermediates between the ORB and the object implementations.
- ▶ Manage the object references and know how to get the corresponding object
- ▶ Support for transparent activation of objects
- ▶ Objects can be transient or persistent
- ▶ **Corba object** = abstract entity that may or may not correspond to some concrete object
- ▶ **IOR** = some binary representation representing a reference to a Corba object
- ▶ **Server** = process where concrete objects (can) run
- ▶ **Servant** = object (instance) in a programming language that implements behaviour for one or more Corba objects
- ▶ **POA** (Portable Object Adapter) is now standard

Object Adapter



POA (1)

- ▶ A POA is an object associated with the ORB
- ▶ There can be more than one POA in the ORB: they form a tree
- ▶ The root POA is called "RootPOA" and can be obtained with `resolve_initial_references`
- ▶ A POA has a number of **policies** that determine how it reacts to requests for Objects
- ▶ E.g. it may have only transient objects or it may be prepared to revive sleeping persistent objects
- ▶ An object reference has the name of the POA and an object key from the POA in its object reference

POA policies

- ▶ ThreadPolicy: ORB_CTRL_MODEL, SINGLE_THREAD_MODEL, MAIN_THREAD_MODEL
- ▶ LifespanPolicy: TRANSIENT, PERSISTENT
- ▶ IdUniquenessPolicy: UNIQUE_ID, MULTIPLE_ID
- ▶ IdAssignmentPolicy: USER_ID, SYSTEM_ID
- ▶ ServantRetentionPolicy (Active Object Map): RETAIN, NON_RETAIN
- ▶ RequestProcessingPolicy: USE_ACTIVE_OBJECT_MAP_ONLY, USE_DEFAULT_SERVANT, USE_SERVANT_MANAGER
- ▶ ImplicitActivationPolicy: IMPLICIT_ACTIVATION, NO_IMPLICIT_ACTIVATION

ThreadPolicy

ORB_CTRL_MODEL	the ORB assigns requests to threads
SINGLE_THREAD_MODEL	no more than one thread may enter any one SINGLE_THREAD_MODEL POA at any given time (multiple requests for this POA will be queued)
MAIN_THREAD_MODEL	all requests in these POA's together will be processed in the main thread (multiple requests for any such POA will be queued)

LifespanPolicy

TRANSIENT	objects created by the POA cannot survive the POA itself: once the POA is destroyed, the objects must cease to exist
PERSISTENT	objects created by the POA may survive the POA: once the POA is destroyed, the objects may or may not cease to exist

IdUniquenessPolicy

UNIQUE_ID	each servant may be associated with no more than a single Objectid value (no more than one Corba object per servant)
MULTIPLE_ID	each servant may be associated with multiple Objectid values (there may be more than one Corba object per servant)

IdAssignmentPolicy

SYSTEM_ID	the ORB assigns Objectid values
USER_ID	the servant assigns Objectid values

ServantRetentionPolicy

RETAIN	the POA maintains an active object map, associating Objectid values with servants; the map can be used to associate a given Objectid with a specific servant
NON_RETAIN	the POA does not maintain an active object map; to find the servant for a given Objectid, the POA either uses a default servant (RequestProcessingPolicy = USE_DEFAULT_SERVANT), or lets a ServantManager locate the servant (RequestProcessingPolicy = USE_SERVANT_MANAGER)

RequestProcessingPolicy

USE_ACTIVE_OBJECT_MAP_ONLY	the POA looks for servants in the active object map
USE_DEFAULT_SERVANT	the POA looks for servants in the active object map (if any); if not found, then the POA forwards the request to the default servant
USE_SERVANT_MANAGER	the POA looks for servants in the active object map (if any); if not found, then the POA has the ServantManager find the servant

ImplicitActivationPolicy

NO_IMPLICIT_ACTIVATION	the POA does not support implicit activation
IMPLICIT_ACTIVATION	the POA supports implicit activation

Activation in POA

- ▶ USE_DEFAULT_SERVANT: POA activates unknown objects by invoking default servant no matter what the object ID is (set with set_servant).
- ▶ RETAIN: servant and object ID are entered into active object map of POA. Activation is done:
 - ▶ explicitly (activate_object or activate_object_with_id)
 - ▶ activate objects on demand: POA invoke a user-supplied servant manager (set with set_servant_manager).
 - ▶ IMPLICIT_ACTIVATION: POA may implicitly activate an object when the server attempts to obtain a reference for a servant that is not already active.
- ▶ NON_RETAIN: POA may use either a default servant or a servant manager to locate an active servant. From the POA's point of view, the servant is active only for the duration of that one request. The servant-object association is not entered into the active object map.

POA (2)

- ▶ The POA Manager is an object that allows you to control the POA's
 - ▶ Activate, deactivate, queue, discard
 - ▶ POAManagerFactory can create POA Managers
- ▶ Adapter Activator can create POA's on demand (based upon Object ID's)
- ▶ A Servant Manager can be written to manage objects that have no servant active
 - ▶ ServantActivator – activates the object at the first request
 - ▶ ServantLocator – creates a servant for each request

POA example

Example in python:

```
# Initialize the ORB
orb = CORBA.ORB_init(sys.argv, CORBA.ORB_ID)
# Find the root POA
poa = orb.resolve_initial_references("RootPOA")

# Create an instance of Hello_impl
hi = Hello_impl()

# Create an object reference,
# implicitly activate the object
ho = hi._this()

# Activate the POA
poaManager = poa._get_the_POAManager()
poaManager.activate()
```

Example ServantActivator

```
class ServantActivator_impl(ServantActivator):
    def incarnate(self, oid, poa):
        hi = Hello_impl()
        return hi
    def etherealize(self, oid, poa, serv,
                    cleanup_in_progress, remaining_activations):
        pass
poa = orb.resolve_initial_references("RootPOA")
poaManager = poa._get_the_POAManager()
poaManager.activate()
child = poa.create_POA("MyPOA", poaManager, policies)

# Create the ServantActivator as the child's ServantManager
sai = ServantActivator_impl()
sao = sai._this()
child.set_servant_manager(sao)
```

Example ServantLocator

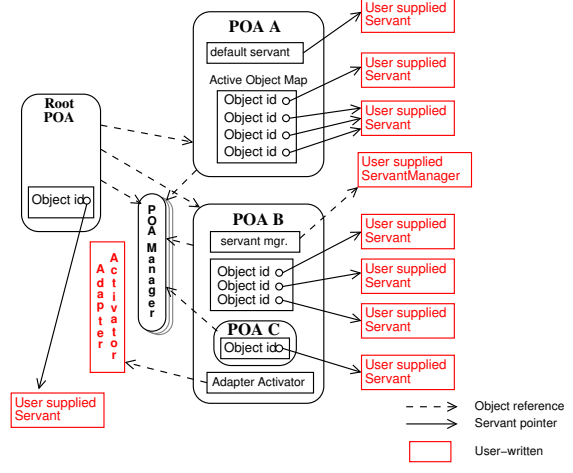
```

class ServantLocator_impl(ServantLocator):
    def preinvoke(self, oid, poa, operation):
        hi = Hello_impl()
        return (hi, "Hmm, cookies")

    def postinvoke(self, oid, poa, operation, cookie, serv):
        pass
    ...
# Create the ServantLocator as the child's ServantManager
sli = ServantLocator_impl()
slo = sli._this()
child.set_servant_manager(slo)

# Create an object reference with no servant
ho = child.create_reference_with_id("MyHello",
                                   CORBA.id(_GlobalIDL.Hello))
print orb.object_to_string(ho)
    
```

POA structure



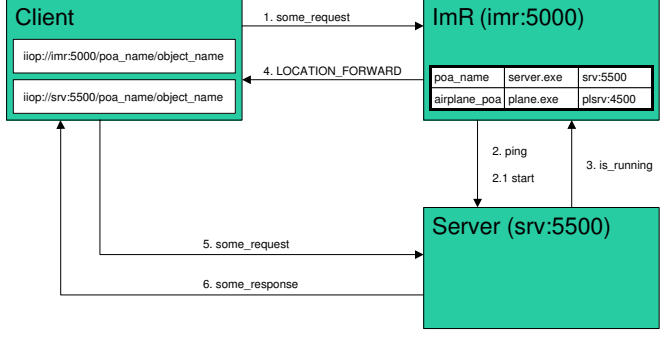
Implementation Repository

- ▶ For persistent objects (existing also when server is not running)
- ▶ For migrating objects
- ▶ No standard interface for Implementation Repository
- ▶ Is intimately tied to a specific ORB
- ▶ It maintains a registry of known servers
- ▶ It records which server is currently running at what host and port number
- ▶ It starts servers on demand if they are registered for automatic activation
- ▶ Should run at a known host/port

Creating Persistent IORs

- ▶ Persistent IORs should contain (a.o):
 - ▶ the repository ID of the most derived interface (as for transient IORs)
 - ▶ the host name and port number of the implementation repository
 - ▶ the object adapter name (embedded in the object key)
 - ▶ the object name (also embedded in the object key)
- ▶ The Implementation Repository maintains connections between Object Adapter names, commands to start a server and hostname/portnumber where server should listen
- ▶ Requests for the object go first to the Implementation Repository
- ▶ After starting the server the Implementation Repository redirects the object id to the real server (location-forward)

Implementation Repository



(source: Douglas C. Schmidt)