Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (01223) 515010**
INTERNATIONAL: **+44 1223 515010**
FAX: **+44 1223 359779**
E-MAIL: **apm@ansa.co.uk**

Training

ANSAwise - Designing Applications with CORBA

Chris Mayers

Abstract

Organizations adopting CORBA technology need to develop an insight into how applications are designed.

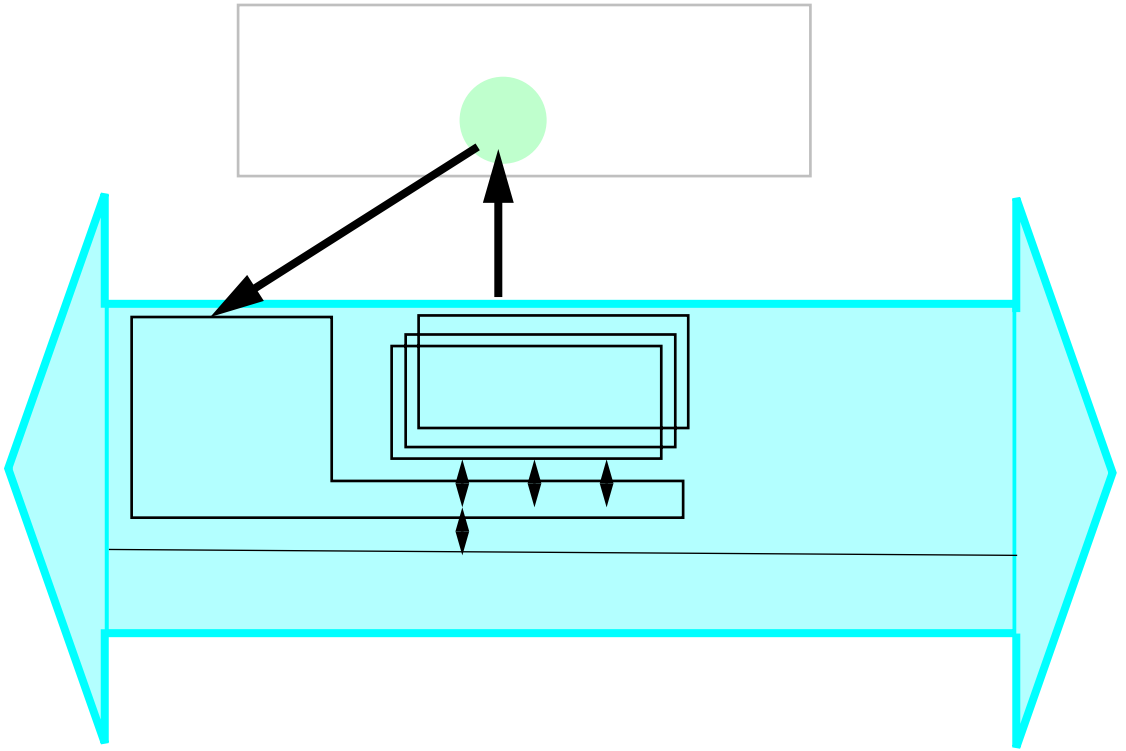
This module of the ANSAwise training programme describes the ORB interfaces (concentrating on the Interface Repository), and also discusses activation policies for servers. It concludes with a simple paper exercise in designing an application based on IDL.

[This variant of APM.1352 merges in information from APM.1345.]

APM.1744.01 **Approved** 2nd April 1996
Briefing Note

Distribution:
Supersedes:
Superseded by:

Designing Applications with CORBA



In this session

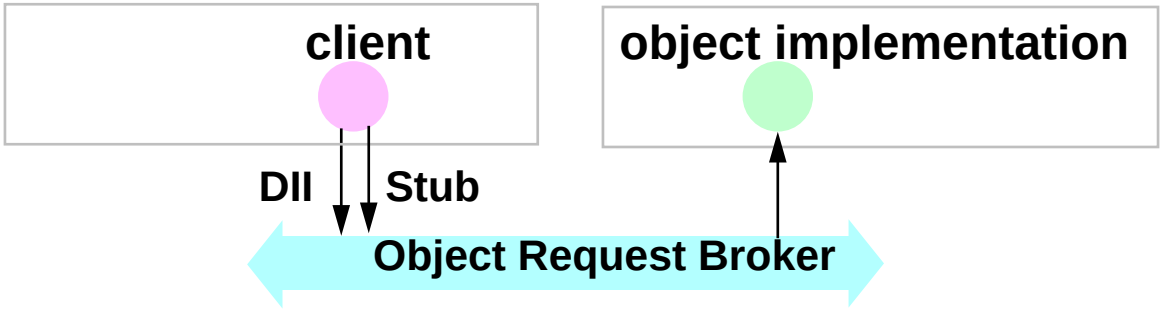
- **Explain in detail some of the standard CORBA interfaces**
  - **including those of the Interface Repository and Implementation Repository**
  - **...as more sophisticated examples of IDL definitions**
- **Explain how to design CORBA object implementations**

CORBA standard interfaces

- **Dynamic Invocation Interface (DII)**
- **ORB Interfaces**
- **Interface Repository and Implementation Repository Interfaces**

Two forms of request

- Clients can make requests via IDL Stubs, or via the Dynamic Invocation Interface (DII)



- There is one IDL Stub per interface
  - but only one DII, which supports all interfaces
- Very roughly speaking
  - requests via IDL stubs are ‘compiled’
  - requests via the DII are ‘interpreted’

IDL Stubs or DII?

- The mappings to IDL Stubs and the DII are the same for all ORBS
  - clients are portable, whether they use IDL Stubs or the DII
- The mapping to IDL Stubs is very different from the mapping to the DII
  - you need to write different source code for the two cases
- Clients can make requests via IDL Stubs for some interfaces, and the DII for others
- Servers support both IDL Stubs and the DII
  - they are not aware of the form of the client request

When you should use the DII

- Application programmers must construct DII parameter lists by hand
- The DII API is large and complex
- There are no checks until run-time
- ... You should therefore only use the DII for writing ORB tools
  - for example, an object browser

Other DII factors

- IDL Stubs only support synchronous requests
  - they do not support ‘deferred synchronous’ requests (initiate/redeem, follow-up RPC)
- The DII cannot be used on pseudo-objects

Interfaces to the ORB itself

- **Two interfaces**
  - **interface ORB**
  - **interface Object**

Interface ORB

- Its specification is

```
interface ORB {
   string object_to_string (in Object obj);
   Object string_to_object (in string str);

   status create_list (
      in long    count,
      out NVList new_list
   );
   status create_operation_list (
      in OperationDef      oper,
      out NVList           new_list
   );
   status get_default_context (out Context ctx);
};
```


Using the ORB interface

- You may need to use object_to_string and string_to_object
  - if you want to store away object references
  - ... in ordinary text files, for instance
- The string form of an object reference is guaranteed to be understood
  - when translated back by string_to_object
- The other operations are mainly for use with the DII

The Object interface

- Its specification is

```
interface Object {
   ImplementationDef get_implementation ();
   InterfaceDef get_interface ();
   boolean is_nil ();
   Object duplicate ();
   void release ();

   status create_request (
      in Context         ctx,
      in Identifier      operation,
      in NVList          arg_list,
      inout NamedValue   result,
      out Request        request,
      in Flags           req_flags
   );
};
```

Using the Object interface

- The Object interface is for manipulating object references
  - is_nil() to check whether an object reference refers to an object
  - duplicate() to duplicate an object reference
  - release() to release an object reference
- If you need a copy of an object reference, you must use duplicate()
  - this allocates storage for the object reference
  - ... it does not copy the object
- Likewise, release() releases storage for the object reference
  - ... it does not destroy the object itself

CORBA Interface Repository

- A CORBA service that provides storage of interface definitions
  - effectively, the logical compiled IDL
- ORBs may need this service themselves
  - to support IDL stubs
  - to support the Dynamic Invocation Interface (DII)
  - to support interoperability between ORBs, and with foreign object systems
- Applications can use this service if they wish
  - for installation of interface definitions
  - for design tools (for example, browsing)
  - for end-user tools (for example, a menu bar constructor)

Structure of the Interface Repository

- The Interface Repository is an ORB Service
  - it has its own IDL
- An ORB may have access to multiple Interface Repositories
- The Interface Repository is not a Trading service
- Not all interfaces are held in the Interface Repository

get_interface() - the key to the Interface Repository

- CORBA does not provide a ‘typeof’ function
- If applications wish to do their own type checking...
  - for example, when retrieving information from the Naming service
  - or for debugging
- ...they can navigate the Interface Repository to retrieve type names, operation signatures, and so on

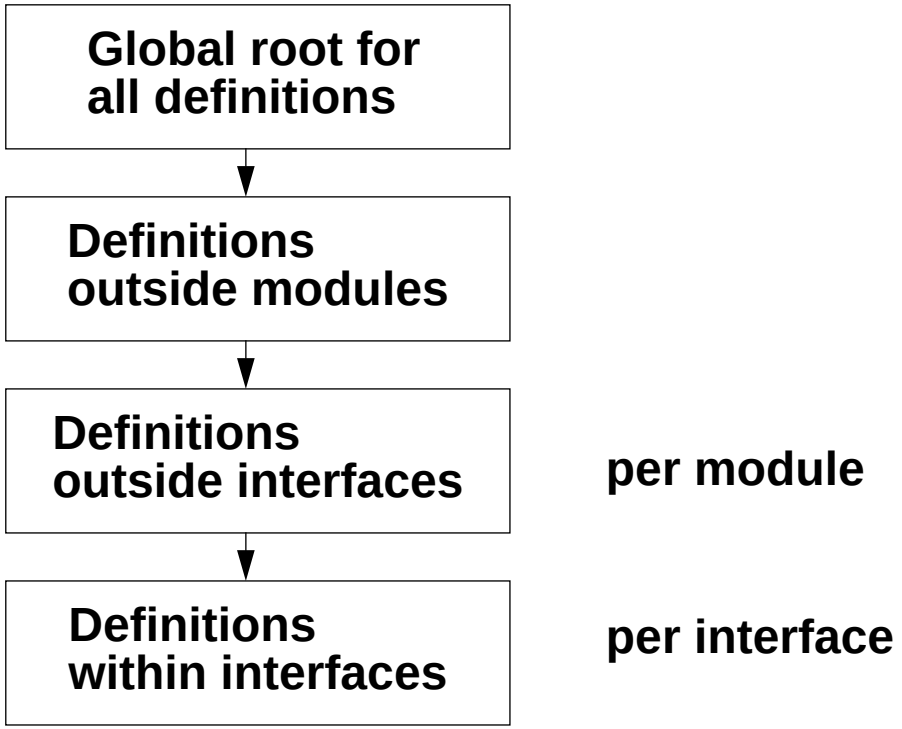
Interface Repository interfaces - generic

- There are two generic interfaces, on which the other Interface Repository interfaces are based
  - interface Contained
  - interface Container
- One interface for the Repository itself
  - interface Repository

Interface Repository interfaces - specific

- ... and one interface for each IDL construct
  - interface ModuleDef
  - Interface InterfaceDef
  - Interface AttributeDef
  - Interface OperationDef
  - Interface ParameterDef
  - Interface TypeDef
  - Interface ConstantDef
  - Interface ExceptionDef
  - Interface TypeCode

Organization of the Interface Repository



Interface Contained

- Note that this uses Container

```
interface Contained {
   attribute Identifier     name;
   attribute RepositoryId   id;
   attribute RepositoryId   defined_in;

   sequence <Container> within ();

   struct Description {
      Identifier               name;
      any                   value;

   };

   Description describe ();
};
```

The any type

- Types in IDL definitions include
  - basic values (like short, boolean)
  - object references (like Container, NamingContext)
- It is sometimes helpful to be able to pass around values that can be either of these
  - as the Interface Repository does here
- There are some restrictions on type any
  - see *CORBA* for details
- Do not recommend that you use type any yourself
  - use type Object (or, better, or more specific object type) instead

Interface Container - 1

- Note that this uses interface Contained

```
interface Container {
   sequence<Contained> contents (
      in InterfaceName limit_type,
      in boolean       exclude_inherited
   );

   sequence <Contained> lookup_name (
      in Identifier    search_name,
      in long          levels_to_search,
      in InterfaceName limit_type,
      in boolean        exclude_inherited
   );
```

Interface Container - 2

```
struct Description{
   Contained          contained_object;
   Identifier         name;
   any                value
};



sequence<Description> describe_contents(
   in InterfaceName limit_type,
   in boolean        exclude_inherited,
   in long           max_return_objs
);

};
```

Interface InterfaceDef

- Note that this is based on both interfaces Container and Contained

```
interface InterfaceDef : Container, Contained {
   sequence<Repositoryid> base_interfaces;

   struct FullInterfaceDescription {
      Identifer                        name;
      RepositoryId                     id;
      RepositoryId                     defined_in;
      sequence<OperationDescription>   operation;
      sequence<AttributeDescription>   attribute;
   };

    FullInterfaceDescription describe_interface();
);
```

CORBA Implementation Repository

- **Holds information about implementations of objects**

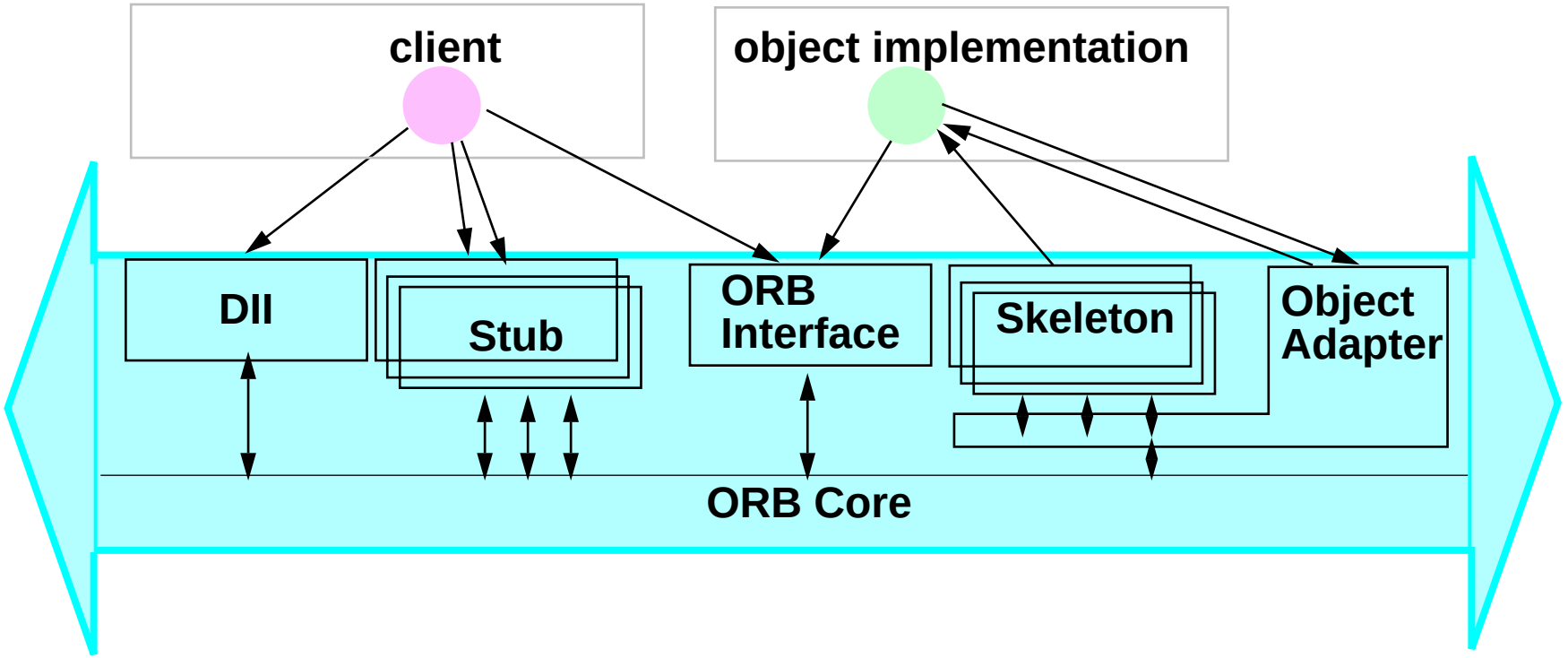
```
interface ImplementationDef {

        // ORB-dependent

};
```

- **You can use get_implementation() as the key to the Implementation Repository**

Structure of the ORB



The structure of an ORB

- One IDL stub and one IDL skeleton per interface
  - automatically generated from IDL
- One DII and one ORB Interface per ORB
- One or more Object Adapters per ORB
- Interfaces with the ORB Core are ORB-dependent

Object Adapters

- Object implementations require general facilities
  - to establish and authenticate object identities
  - to create and destroy objects
  - to access ORB-dependent facilities
- This is done via an Object Adapter
  - the object adapter is also used implicitly by skeletons
- There is always a Basic Object Adapter (BOA); there may be others
  - for example, for legacy systems integration, or for high-performance implementations (as libraries, or as OO databases)
  - the object implementation can choose which Object Adapter to use

Basic Object Adapter

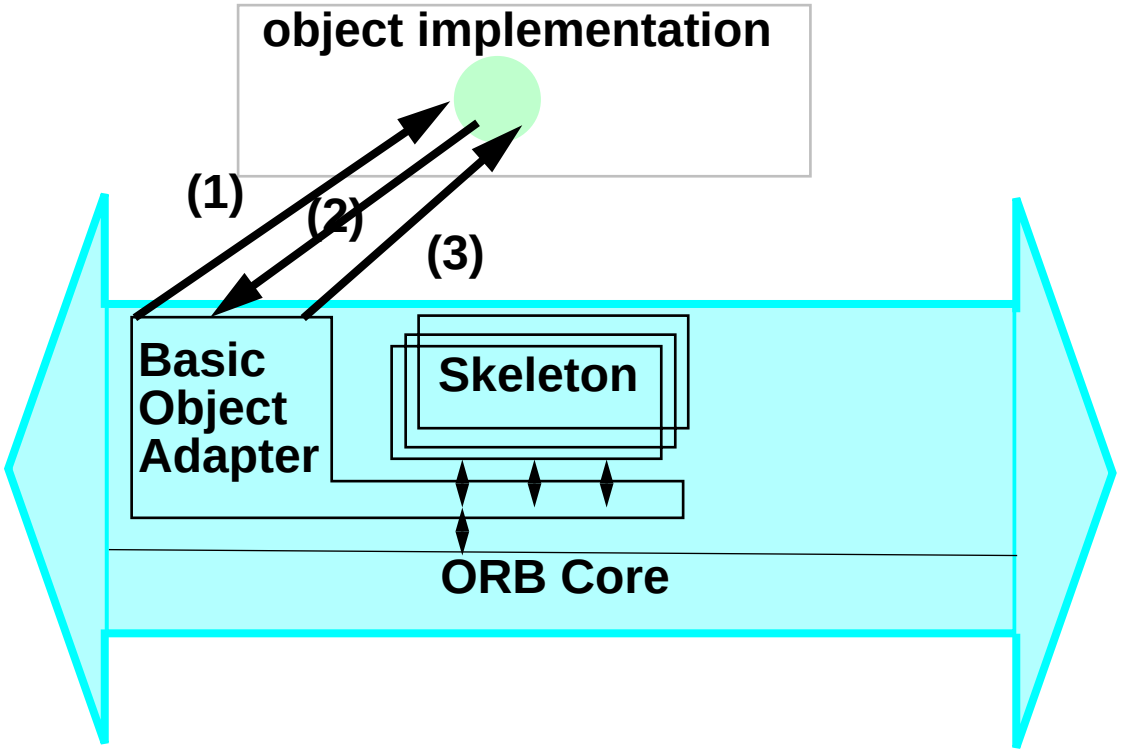
- This has an IDL specification of its own
  - interface BOA
- This is likely to be hard-wired and pre-linked into the ORB

Basic Object Adapter

- Calls flow in both directions between the Basic Object Adapter and the implementation
  - (down) calls from the object implementation to the BOA
  - upcalls from the BOA to the object implementation
- Calls from object implementation use the BOA interface
  - an IDL definition, in the usual way
- Upcalls from the BOA use the language mapping
- Object implementations must be involved in both calls and upcalls
  - to implement the operations of their interfaces
  - during activation of implementation and objects

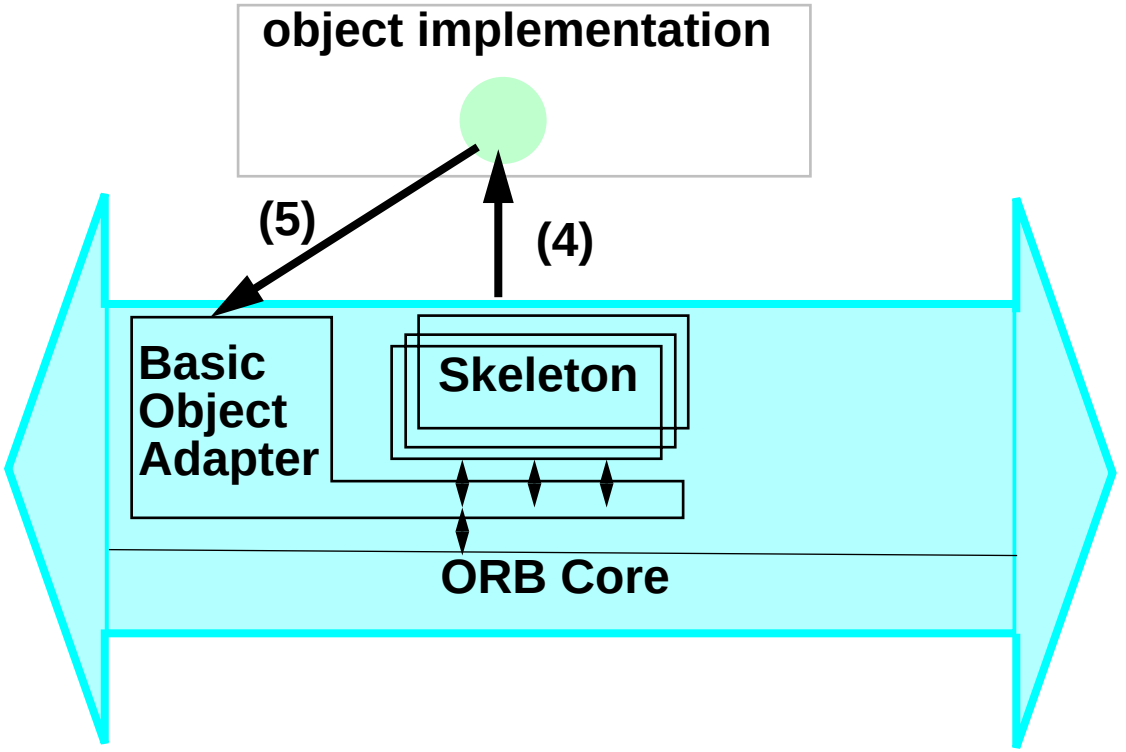
How the Basic Object Adapter works - activation

- Activate Implementation, Register Implementation, Activate Object...



How the Basic Object Adapter works - use

- ... Invoke Method, Access BOA Service



Object Implementations

- **Object Implementations do not have to be traditional programs**
  - **they can be scripts, loadable modules, or any other kind of executable software**
- **Object Implementations can be connected to the ORB in different ways**
  - **this is ORB-dependent**

Activation policies

- CORBA allows a variety of activation policies
  - important when servers support multiple objects
- The Basic Object Adapter supports four policies
  - shared server: multiple active objects sharing the same implementation
  - unshared server: one object per implementation
  - server-per-method: each invocation activates a new server - the server terminates when the invocation's method completes
  - persistent server: the server is activated by some external mechanism - it must be available before invocations can be processed

Example - the Daemon Game

- A simple game having several players
  - the players are part of the system design
- In the system there is a daemon that generates Bump signals randomly
  - a player has to guess whether the number of generated Bump signals is even or odd
- The guess is made by sending a Probe signal to the system
  - which responds with either Win or Lose

The Daemon game - continued

- The system keeps track of the score of each player
  - a player can ask for their current score by sending the Score signal
- Before playing, a player must log in by using Newgame
  - and afterwards log out using Endgame
- A player can play more than one game at the same time

Designing the Daemon game

- Decide how many interfaces are necessary
- Draft the IDL for the interfaces
- Suggest an activation policy for the object implementations
- Consider any other issues with this design

Get ready to discuss this

Your notes

Summary

- Clients and object implementations can use the ORB standard interfaces as necessary
- Object implementations must use the BOA interface
- For more on this topic
  - on the CORBA Interface Repository and Implementation Repository, see *CORBA* (OMG)
  - on the Basic Object Adapter, also see this document
  - for a discussion of the Daemon Game example, see ISO/IEC TR 10167

ORB Implementations

- The CORBA Architecture is intended to allow a variety of ORB implementations, for example
  - static libraries linked into clients and servers
  - dynamic-linked libraries bound to clients and servers at run time
  - as a centralized server
  - built into the underlying operating system
- The ORB implementations affect the way applications are built and installed...
  - ... but applications are still source-code portable and interoperable

Multiple ORB Implementations

- An ORB installation may consist of several interoperating ORBs
  - supplied by different vendors
  - implemented in different ways
- These ORB implementations may use
  - different representations for object references
  - different means for performing invocations
- These differences are transparent to the application
  - the ORB implementations must be capable of resolving these differences automatically, for interoperability

ORB Integration with other object systems

- As well as wrapping non-OO applications, an ORB can integrate with non-CORBA object systems
- This can be done in these ways