CS 5523 Lecture 9: CORBA

- *Discuss Laboratory 2*
- *CORBA objects and IDL*
- *The ShapeList example in CORBA*
- *CORBA naming service*
- *Other CORBA services*
- *Recommended reading*

CORBA overview:

- *Middleware that allows communication between programs independent of language, OS, hardware, and network*
- *Applications are built from CORBA objects*
- *CORBA objects implement interfaces defined in IDL*
- *Clients access methods in the IDL interfaces by RMI*
- *RMI is implemented by an ORB (Object Request Broker)*

Remote interfaces – Java RMI versus CORBA:

- *CORBA – uses IDL to specify remote interfaces*
- *JAVA – uses ordinary interfaces that are extended by the keyword* `remote`*.*

CORBA objects:

- *implement an IDL interface*
- *have a remote object reference*
- *can respond to invocations of methods in the IDL interface*

How do CORBA objects differ from Java RMI?

- *CORBA objects can be implemented in non-OO languages*
- *clients don’t have to be objects*
- *classes cannot be implemented in IDL – so no objects can be passed, only data structures*

How does a data structure differ from an object?

CORBA IDL interfaces:

- *specify a name and a set of methods*
- *parameters are marked with keywords in, out, or inout*
- *parameters can be of a primitive type or constructed type*
- *allows exceptions to be defined in interfaces and thrown by methods*
- *invocation is at-most-once by default (can also specify oneway)*

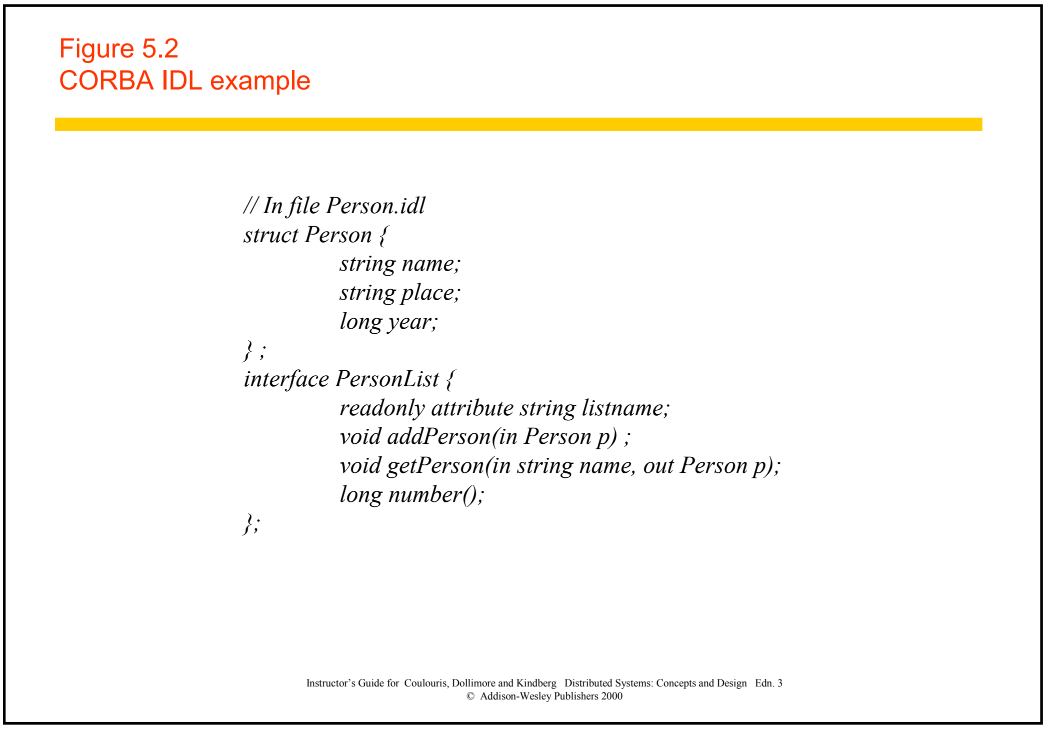


Figure 5.2
CORBA IDL example

```
// In file Person.idl
struct Person {
            string name;
            string place;
            long year;
} ;
interface PersonList {
            readonly attribute string listname;
            void addPerson(in Person p) ;
            void getPerson(in string name, out Person p);
            long number();
};
```

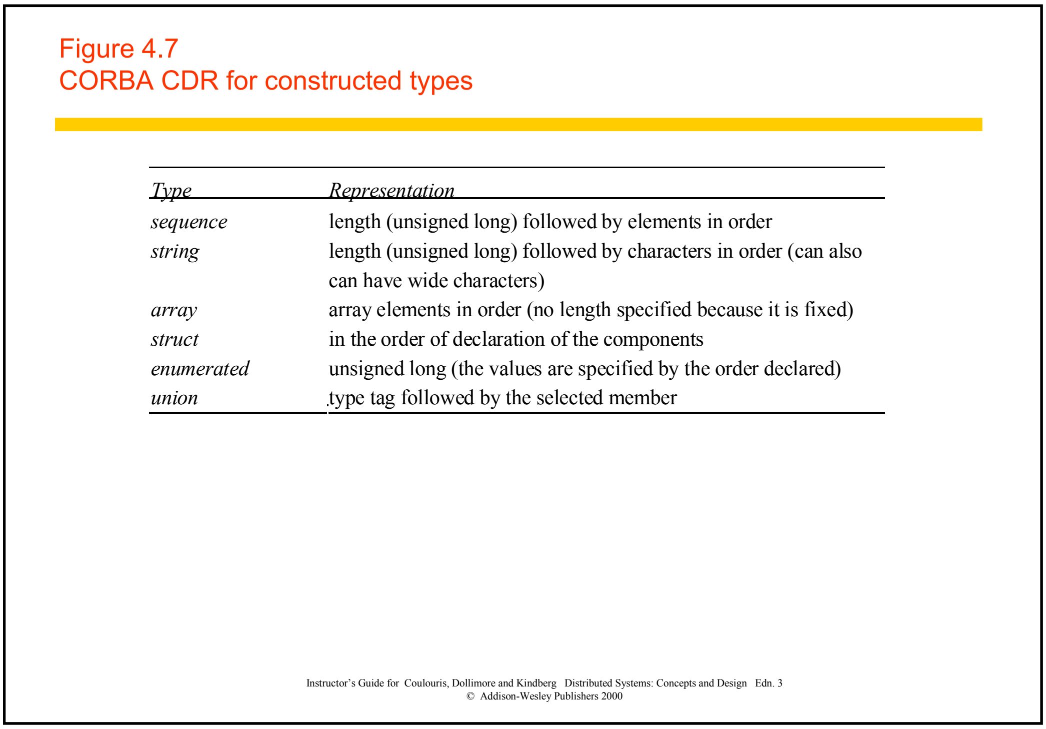


Figure 4.7
CORBA CDR for constructed types

Type	*Representation*
sequence	length (unsigned long) followed by elements in order
string	length (unsigned long) followed by characters in order (can also can have wide characters)
array	array elements in order (no length specified because it is fixed)
struct	in the order of declaration of the components
enumerated	unsigned long (the values are specified by the order declared)
union	type tag followed by the selected member

Figure 4.8
CORBA CDR message

index in sequence of bytes	4 bytes	*notes on representation*
0–3	5	*length of string*
4–7	"Smit"	*'Smith'*
8–11	"h___"	
12–15	6	*length of string*
16–19	"Lond"	*'London'*
20-23	"on__"	
24–27	1934	*unsigned long*

The flattened form represents a *Person* struct with value: {'Smith', 'London', 1934}

Figure 17.1
IDL interfaces Shape and ShapeList

```
struct Rectangle{                          1
      long width;
      long height;
      long x;
      long y;
} ;

struct GraphicalObject {                   2
      string type;
      Rectangle enclosing;
      boolean isFilled;
};

interface Shape {                                                          3
      long getVersion() ;
      GraphicalObject getAllState() ;       // returns state of the GraphicalObject
};

typedef sequence <Shape, 100> All;                                         4
interface ShapeList {                                                      5
      exception FullException{ };                                          6
      Shape newShape(in GraphicalObject g) raises (FullException);         7
      All allShapes();                      // returns sequence of remote object references   8
      long getVersion() ;
};
```

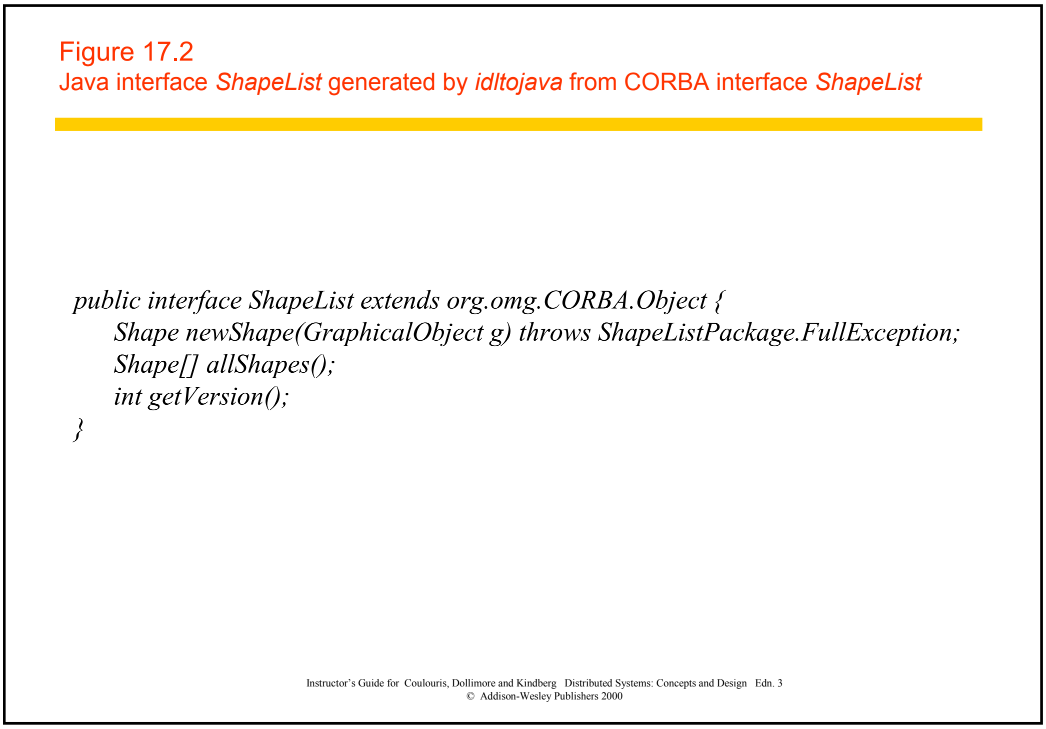


Figure 17.2
Java interface *ShapeList* generated by *idltojava* from CORBA interface *ShapeList*

```
public interface ShapeList extends org.omg.CORBA.Object {
    Shape newShape(GraphicalObject g) throws ShapeListPackage.FullException;
    Shape[] allShapes();
    int getVersion();
}
```

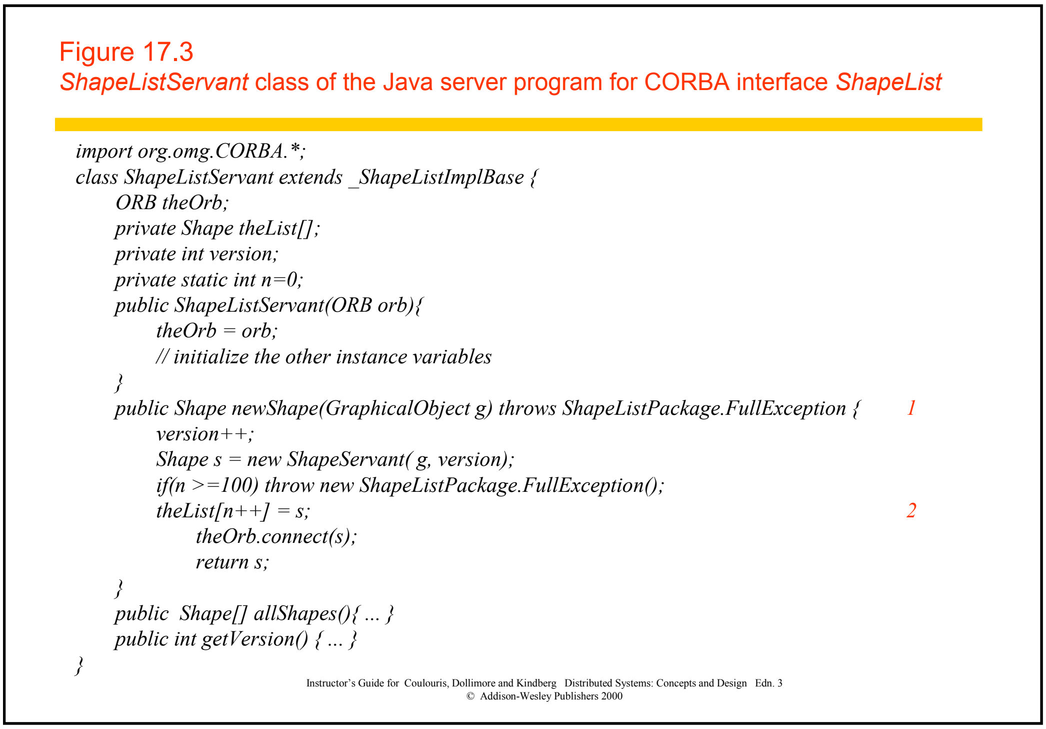


Figure 17.3
ShapeListServant class of the Java server program for CORBA interface *ShapeList*

```
import org.omg.CORBA.*;
class ShapeListServant extends _ShapeListImplBase {
    ORB theOrb;
    private Shape theList[];
    private int version;
    private static int n=0;
    public ShapeListServant(ORB orb){
        theOrb = orb;
        // initialize the other instance variables
    }
    public Shape newShape(GraphicalObject g) throws ShapeListPackage.FullException {      1
        version++;
        Shape s = new ShapeServant( g, version);
        if(n >=100) throw new ShapeListPackage.FullException();
        theList[n++] = s;                                                                   2
            theOrb.connect(s);
            return s;
    }
    public  Shape[] allShapes(){ ... }
    public int getVersion() { ... }
}
```

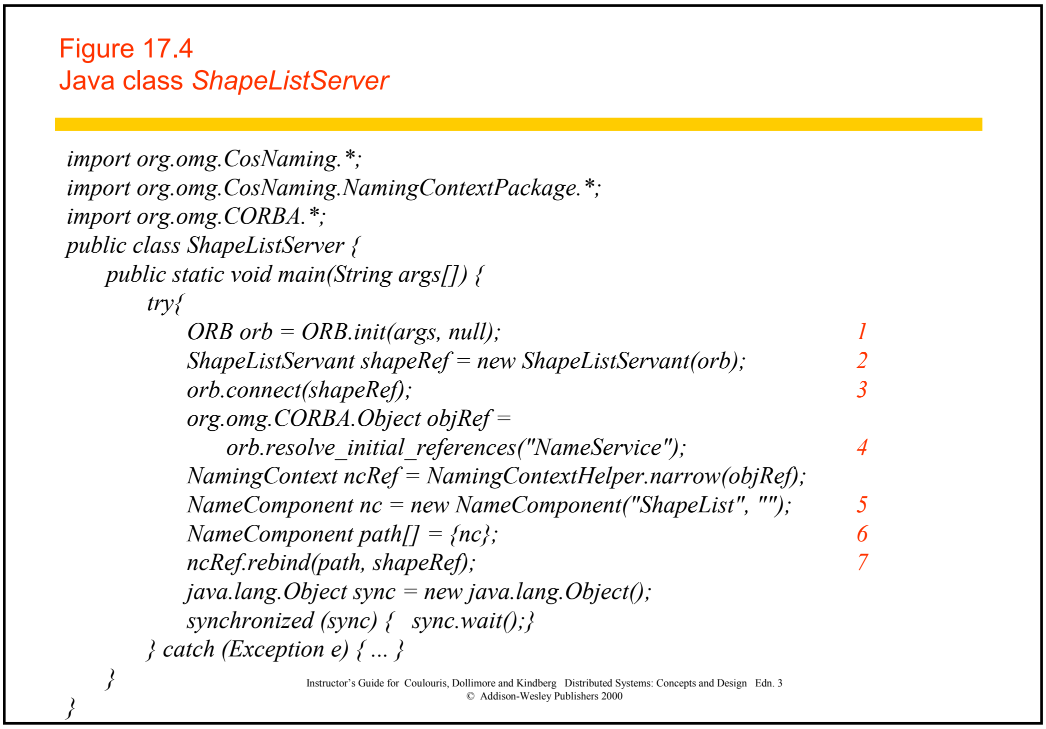


Figure 17.4
Java class *ShapeListServer*

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class ShapeListServer {
    public static void main(String args[]) {
        try{
            ORB orb = ORB.init(args, null);                                  1
            ShapeListServant shapeRef = new ShapeListServant(orb);           2
            orb.connect(shapeRef);                                           3
            org.omg.CORBA.Object objRef =
                orb.resolve_initial_references("NameService");               4
            NamingContext ncRef = NamingContextHelper.narrow(objRef);
            NameComponent nc = new NameComponent("ShapeList", "");           5
            NameComponent path[] = {nc};                                     6
            ncRef.rebind(path, shapeRef);                                    7
            java.lang.Object sync = new java.lang.Object();
            synchronized (sync) {   sync.wait();}
        } catch (Exception e) { ... }
    }
}
```

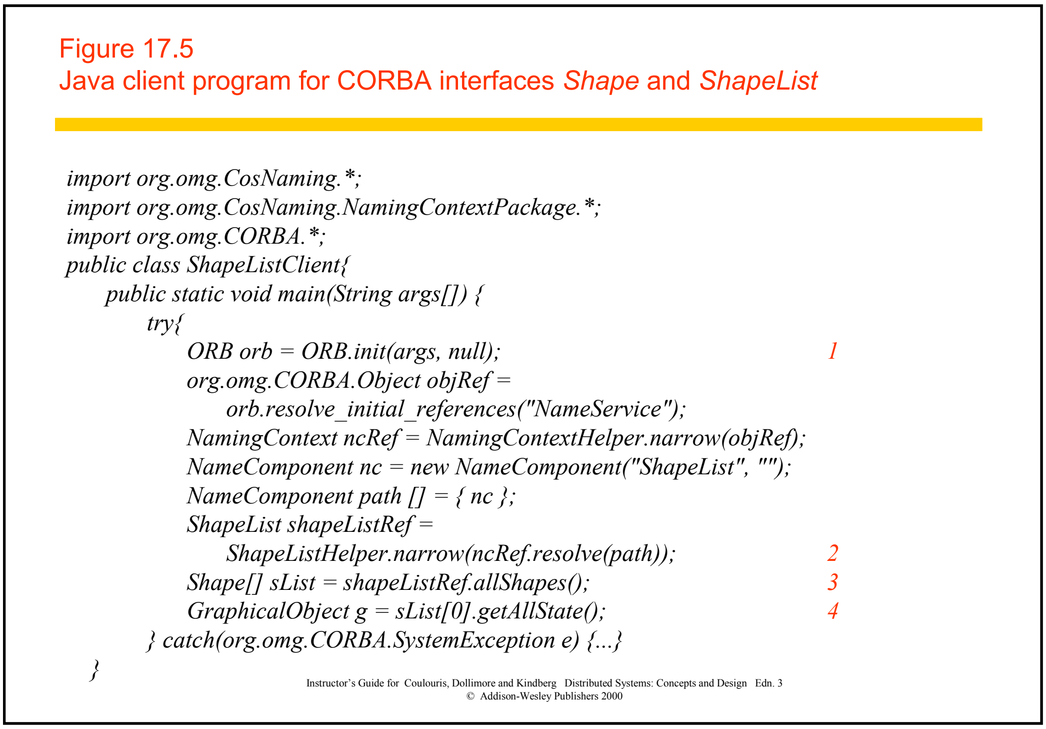


Figure 17.5
Java client program for CORBA interfaces *Shape* and *ShapeList*

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class ShapeListClient{
    public static void main(String args[]) {
        try{
            ORB orb = ORB.init(args, null);                               1
            org.omg.CORBA.Object objRef =
                orb.resolve_initial_references("NameService");
            NamingContext ncRef = NamingContextHelper.narrow(objRef);
            NameComponent nc = new NameComponent("ShapeList", "");
            NameComponent path [] = { nc };
            ShapeList shapeListRef =
                ShapeListHelper.narrow(ncRef.resolve(path));              2
            Shape[] sList = shapeListRef.allShapes();                     3
            GraphicalObject g = sList[0].getAllState();                   4
        } catch(org.omg.CORBA.SystemException e) {...}
  }
```

Figure 17.6
The main components of the CORBA architecture

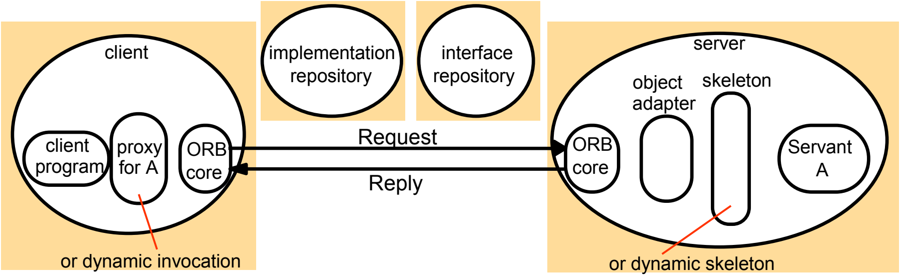


Figure 17.7
IDL module Whiteboard

```
module Whiteboard {
    struct Rectangle{
    ...} ;
    struct GraphicalObject {
    ...};
    interface Shape {
    ...};
    typedef sequence <Shape, 100> All;
    interface ShapeList {
    ...};
};
```

Figure 17.8
IDL constructed types – 1

Type	*Examples*	*Use*
sequence	*typedef sequence <Shape, 100> All;* *typedef sequence <Shape> All* bounded and unbounded sequences of *Shapes*	Defines a type for a variable-length sequence of elements of a specified IDL type. An upper bound on the length may be specified.

string	*String name;*
	typedef string<8> SmallString;
	unbounded and bounded sequences of characters
		Defines a sequences of characters, terminated by the null character. An upper bound on the length may be specified.

array	*typedef octet uniqueId[12];* *typedef GraphicalObject GO[10][8]*	Defines a type for a multi-dimensional fixed-length sequence of elements of a specified IDL type.

this figure continues on the next slide

Figure 17.8
IDL constructed types – 2

Type	*Examples*	*Use*

record	*struct GraphicalObject {*
	string type;
	Rectangle enclosing;
	boolean isFilled;
	};
		Defines a type for a record containing a group of related entities. *Structs* are passed by value in arguments and results.

enumerated	*enum Rand* *(Exp, Number, Name);*	The enumerated type in IDL maps a type name onto a small set of integer values.

union	*union Exp switch (Rand) {*
	case Exp: string vote;
	case Number: long n;
	case Name: string s;
	};
		The IDL discriminated union allows one of a given set of types to be passed as an argument. The header is parameterized by an *enum*, which specifies which member is in use.

CORBA pseudo objects:

- *provide interfaces to the functionality of the ORB*
- *have IDL interfaces, but cannot be passed as remote references*
- *examples:*
  - *init – method to initialize the ORB*
  - *connect – method used to register objects with the ORB*

Page 684
CORBA interoperable object references

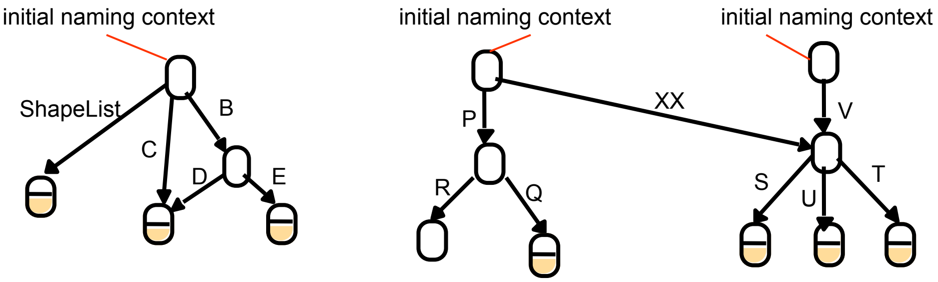
IOR format

IDL interface type name	Protocol and address details			Object key	
interface repository identifier	IIOP	host domain name	port number	adapter name	object name

CORBA naming service:

- *binder providing facilities for servers to register remote objects*
- *provides facilities for clients to resolve names by name*
- *names are structured hierarchically*
- *each name in a path is inside a structure NameComponent*

Figure 17.9
Naming graph in CORBA Naming Service

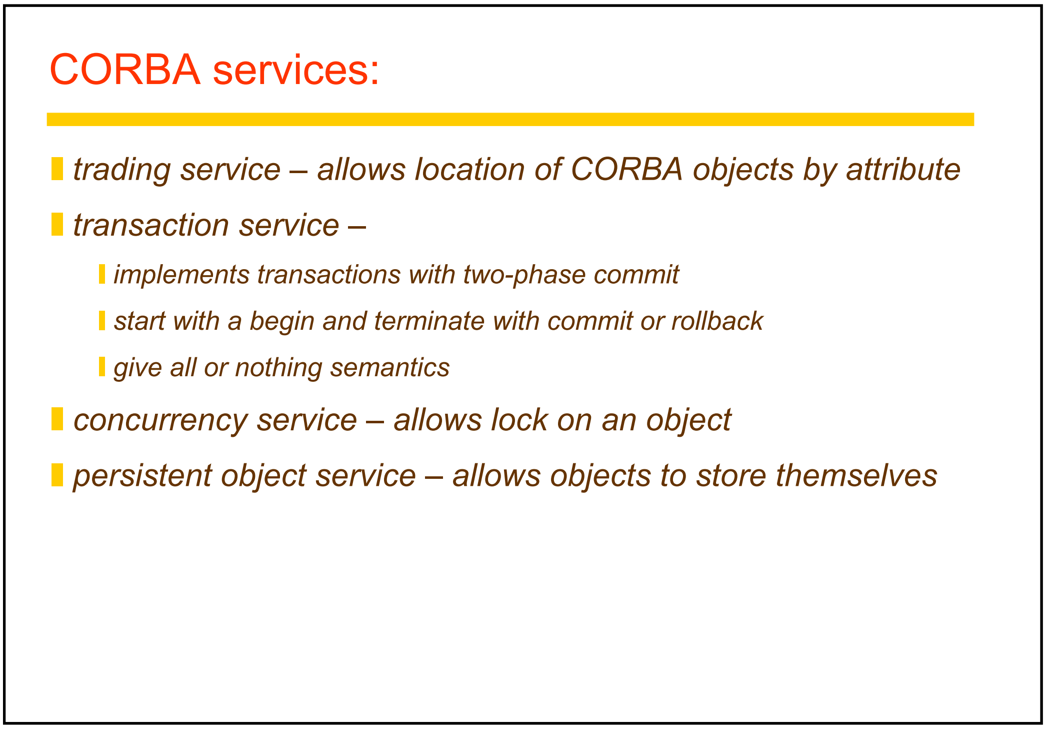


CORBA naming service (contined):

- initial naming context – provides a root for a set of bindings
- clients and servers request initial naming context
- an object of type NamingContext is returned and names are relative to it
- an object is either a remote object or a NamingContext
- names are of type NameComponents and have a name and a kind.
- a Name type is a sequence of NameComponents

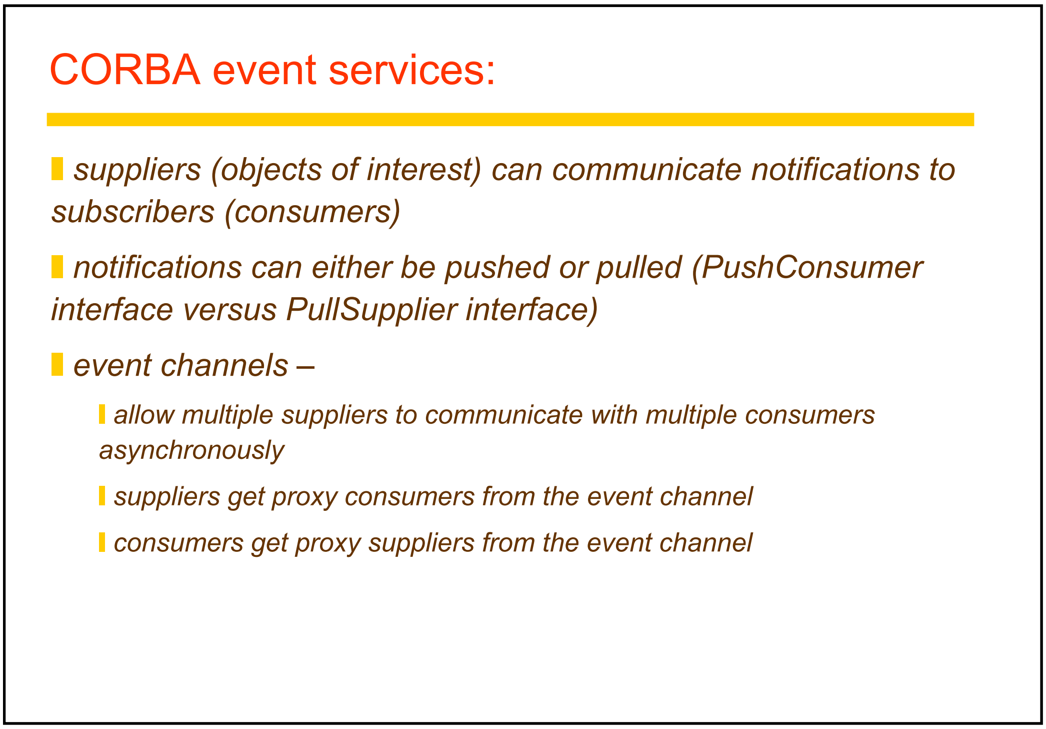
Figure 17.10
Part of the CORBA Naming Service NamingContext interface in IDL

```
struct NameComponent { string id; string kind; };

typedef sequence <NameComponent> Name;

interface NamingContext  {
    void bind (in Name n,  in Object  obj);
        binds the given name and remote object reference in my context.
    void unbind (in Name n);
        removes an existing binding with the given name.
    void bind_new_context(in Name n);
        creates a new naming context and binds it to a given name in my context.
    Object  resolve (in Name n);
         looks up the name in my context and returns its remote object reference.
    void list  (in unsigned long how_many, out BindingList bl, out BindingIterator bi);
        returns the names in the bindings in my context.
};
```



CORBA services:

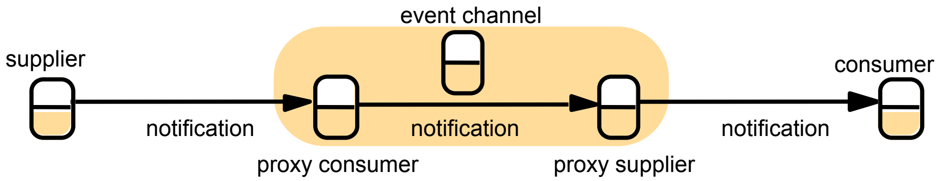
- *trading service – allows location of CORBA objects by attribute*
- *transaction service –*
  - *implements transactions with two-phase commit*
  - *start with a begin and terminate with commit or rollback*
  - *give all or nothing semantics*
- *concurrency service – allows lock on an object*
- *persistent object service – allows objects to store themselves*



CORBA event services:

- *suppliers (objects of interest) can communicate notifications to subscribers (consumers)*
- *notifications can either be pushed or pulled (PushConsumer interface versus PullSupplier interface)*
- *event channels –*
  - *allow multiple suppliers to communicate with multiple consumers asynchronously*
  - *suppliers get proxy consumers from the event channel*
  - *consumers get proxy suppliers from the event channel*

Figure 17.11
CORBA event channels



CORBA notification services:

- *extends the event server*
- *notifications may be data structures*
- *event consumers may use filters*
- *event suppliers can discover which events consumers are interested in*
- *channel properties can be configured*
- *an event repository is provided*

CORBA recommended reading:

The October 1998 Issue of the Communications of the ACM was devoted to new developments in CORBA. It contains many excellent articles.

For next time:

- *Answer questions 5.1, 5.2, 5.3, 5.4, 5.5 and 5.12*
- *Read CDK 6.1-6.3*