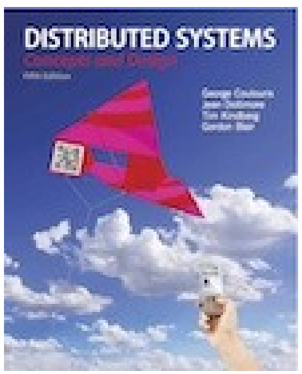
# Slides for Chapter 8:
# Distributed Objects and Components




*From* **Coulouris, Dollimore, Kindberg and Blair**

**Distributed Systems: Concepts and Design**

# Figure 8.1
# Distributed objects

| *Objects* | *Distributed objects* | *Description of distributed object* |
|---|---|---|
| Object references | Remote object references | Globally unique reference for a distributed object; may be passed as a parameter. |
| Interfaces | Remote interfaces | Provides an abstract specification of the methods that can be invoked on the remote object; specified using an interface definition language (IDL). |
| Actions | Distributed actions | Initiated by a method invocation, potentially resulting in invocation chains; remote invocations use RMI. |
| Exceptions | Distributed exceptions | Additional exceptions generated from the distributed nature of the system, including message loss or process failure. |
| Garbage collection | Distributed garbage collection | Extended scheme to ensure that an object will continue to exist if at least one object reference or remote object reference exists for that object, otherwise, it should be removed. Requires a distributed garbage collection algorithm. |

# Figure 8.2
# 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 8.3
# IDL module Whiteboard

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

# Figure 8.4
# IDL constructed types – 1

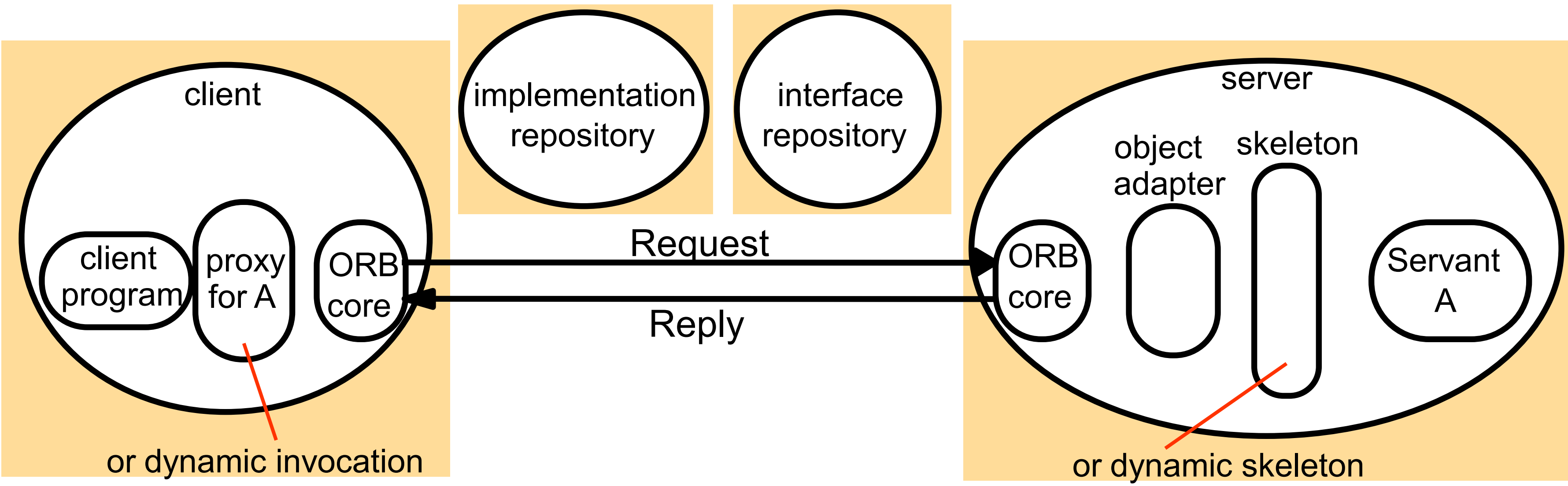
| *Type* | *Examples* | *Use* |
|---|---|---|
| *sequence* | *typedef sequence <Shape, 100> All;*<br>*typedef sequence <Shape> All*<br>bounded and unbounded sequences of *Shapes* | Defines a type for a variable-length sequence of elements of a specified IDL type. An upper bound on the length may be specified. |
| *string* | *String name;*<br>*typedef string<8> SmallString;*<br>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];*<br>*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 8.4
# IDL constructed types – 2

| *Type* | *Examples* | *Use* |
|---|---|---|
| *record* | *struct GraphicalObject {*<br>*string type;*<br>*Rectangle enclosing;*<br>*boolean isFilled;*<br>*};* | Defines a type for a record containing a group of related entities. *Structs* are passed by value in arguments and results. |
| *enumerated* | *enum Rand*<br>*(Exp, Number, Name);* | The enumerated type in IDL maps a type name onto a small set of integer values. |
| *union* | *union Exp switch (Rand) {*<br>*case Exp: string vote;*<br>*case Number: long n;*<br>*case Name: string s;*<br>*};* | 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. |

# Figure 8.5
## The main components of the CORBA architecture



# Figure 8.6
# CORBA Services (1)

| *CORBA Service* | *Role* | *Further details* |
|---|---|---|
| *Naming service* | Supports naming in CORBA, in particular mapping names to remote object references within a given naming context (see Chapter 9). | [OMG 2004b] |
| *Trading service* | Whereas the Naming service allows objects to be located by name, the Trading service allows them to be located by attribute; that is, it is a directory service. The underlying database manages a mapping of service types and associated attributes onto remote object references. | [OMG 2000a, Henning and Vinoski 1999] |
| *Event service* | Allows objects of interest to communicate notifications to subscribers using ordinary CORBA remote method invocations (see Chapter 6 for more on event services generally). | [Farley 1998, OMG 2004c] |
| *Notification service* | Extends the event service with added capabilities including the ability to define filters expressing events of interest and also to define the reliability and ordering properties of the underlying event channel. | [OMG 2004d] |

this figure continues on the next slide

# Figure 8.6
# CORBA Services (continued)

| | | |
|---|---|---|
| *Security service* | Supports a range of security mechanisms including authentication, access control, secure communication, auditing and nonrepudiation (see Chapter 11). | [Blakely 1999, Baker 1997, OMG 2002b] |
| *Transaction service* | Supports the creation of both flat and nested transactions (as defined in Chapters 16 and 17). | [OMG 2003] |
| *Concurrency control service* | Uses locks to apply concurrency control to the access of CORBA objects (may be used via the transaction service or as an independent service). | [OMG 2000b] |
| *Persistent state service* | Offers a persistent object store for CORBA, used to save and restore the state of CORBA objects (implementations are retrieved from the implementation repository). | [OMG 2002d] |
| *Lifecycle service* | Defines conventions for creating, deleting, copying and moving CORBA objects; for example, how to use factories to create objects. | [OMG 2002e] |

# Figure 8.7
## Java interfaces generated by *idlj* from CORBA interface *ShapeList*

```
public interface ShapeListOperations {
    Shape newShape(GraphicalObject g) throws ShapeListPackage.FullException;
    Shape[] allShapes();
    int getVersion();
}

public interface ShapeList extends ShapeListOperations, org.omg.CORBA.Object,
            org.omg.CORBA.portable.IDLEntity { }
```

## Figure 8.8
*ShapeListServant* class of the Java server program for CORBA interface *ShapeList*

```
import org.omg.CORBA.*;
import org.omg.PortableServer.POA;
class ShapeListServant extends ShapeListPOA {
    private POA theRootpoa;
    private Shape theList[];
    private int version;
    private static int n=0;
    public ShapeListServant(POA rootpoa){
        theRootpoa = rootpoa;
        //  initialize the other instance variables
    }
// continued on the next slide
```

# Figure 8.8 continued

```
public Shape newShape(GraphicalObject g)
        throws ShapeListPackage.FullException {                        1
        version++;
        Shape s = null;
        ShapeServant shapeRef = new ShapeServant( g, version);
        try {
            org.omg.CORBA.Object ref =
                theRoopoa.servant_to_reference(shapeRef);              2
            s = ShapeHelper.narrow(ref);
        } catch (Exception e) {}
         if(n >=100) throw new ShapeListPackage.FullException();
        theList[n++] = s;
        return s;
    }
    public  Shape[] allShapes(){ ... }
    public int getVersion() { ... }
}
```

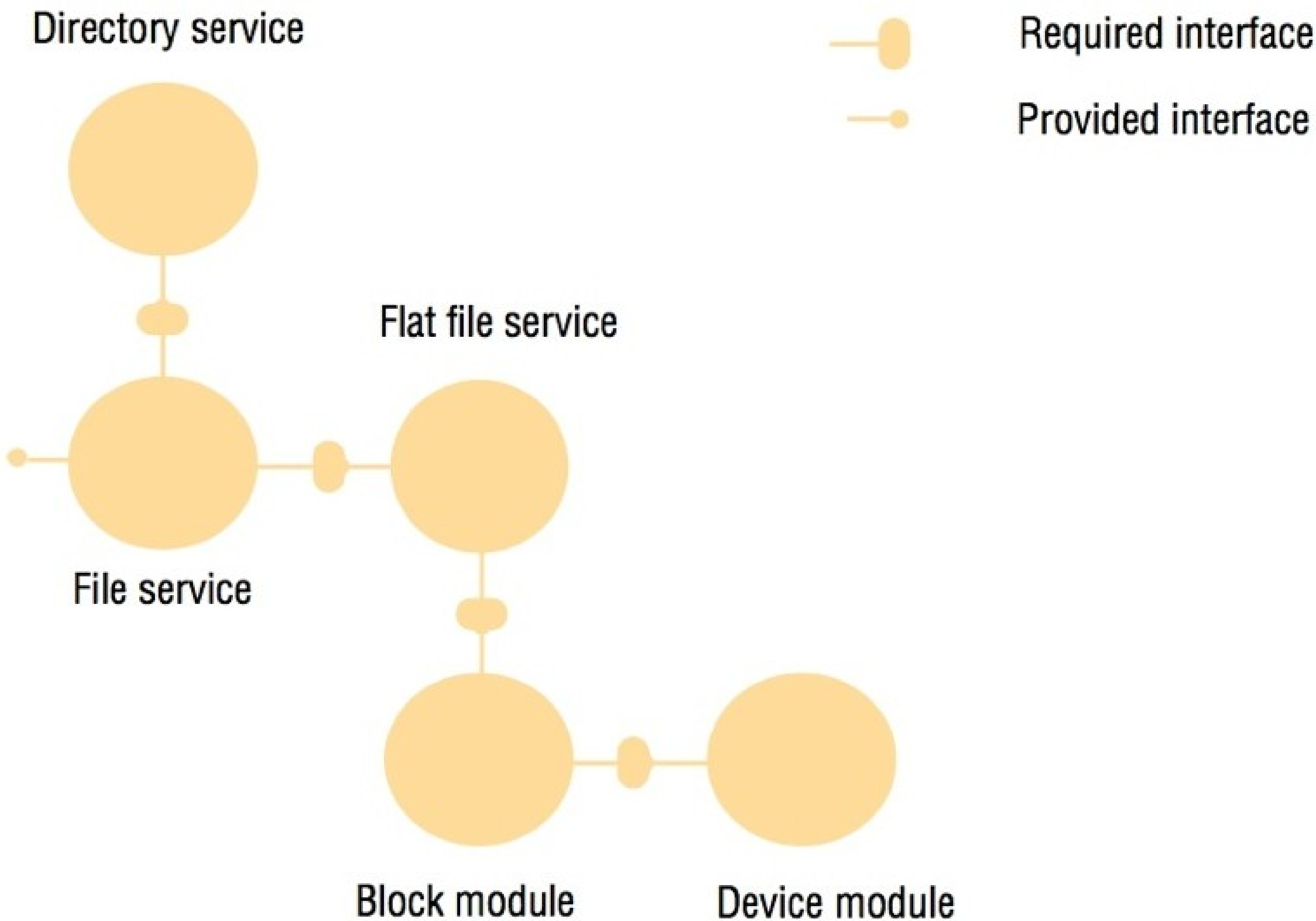
# Figure 8.9
# Java class *ShapeListServer*

```
import org.omg.CosNaming.*;   import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;      import org.omg.PortableServer.*;
public class ShapeListServer {
    public static void main(String args[]) {
      try{
        ORB orb = ORB.init(args, null);                                              1
        POA rootpoa =  POAHelper.narrow(orb.resolve_initial_references("RootPOA"));2
        rootpoa.the_POAManager().activate();                                         3
        ShapeListServant SLSRef = new ShapeListServant(rootpoa);                     4
        org.omg.CORBA.Object ref = rootpoa.servant_to_reference(SLSRef);             5
        ShapeList SLRef = ShapeListHelper.narrow(ref);
        org.omg.CORBA.Object objRef =orb.resolve_initial_references("NameService");
        NamingContext ncRef = NamingContextHelper.narrow(objRef);                    6
        NameComponent nc = new NameComponent("ShapeList", "");                       7
        NameComponent path[] = {nc};                                                 8
        ncRef.rebind(path, SLRef);                                                   9
        orb.run();                                                                   10
      } catch (Exception e) { ... }
    }}
```

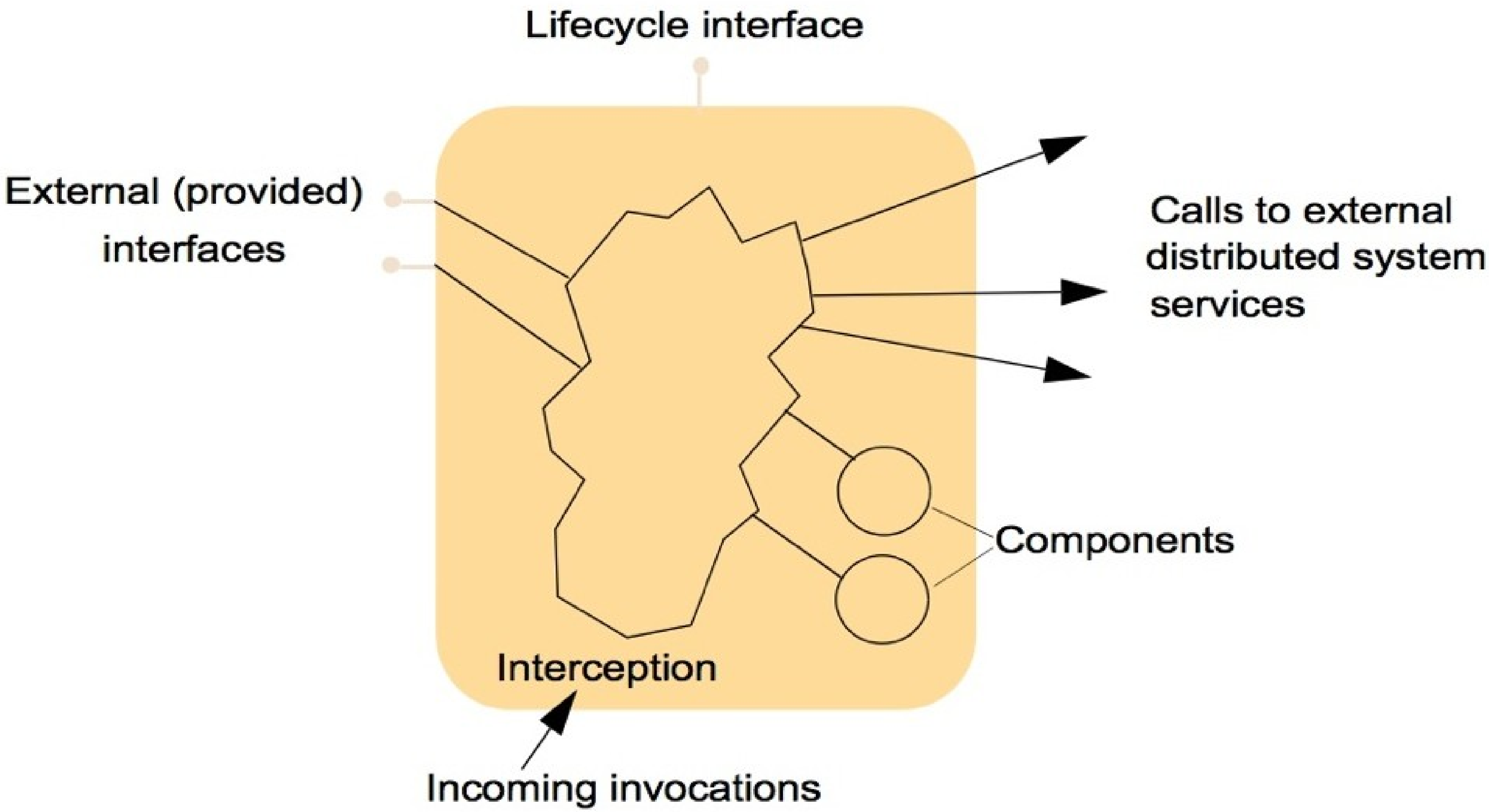
# Figure 8.10
## Java client program for CORBA interfaces *Shape* and *ShapeList*

```
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 8.11
# An example software architecture



# Figure 8.12
# The structure of a container



# Figure 8.13
# Application servers

| *Technology* | *Developed by* | *Further details* |
|---|---|---|
| *WebSphere Application Server* | IBM | [www.ibm.com] |
| *Enterprise JavaBeans* | SUN | [java.sun.com XII] |
| *Spring Framework* | SpringSource (a division of VMware) | [www.springsource.org] |
| *JBoss* | JBoss Community | [www.jboss.org] |
| *CORBA Component Model* | OMG | [Wang *et al.* 2001] |
| *JOnAS* | OW2 Consortium | [jonas.ow2.org] |
| *GlassFish* | SUN | [glassfish.dev.java.net] |

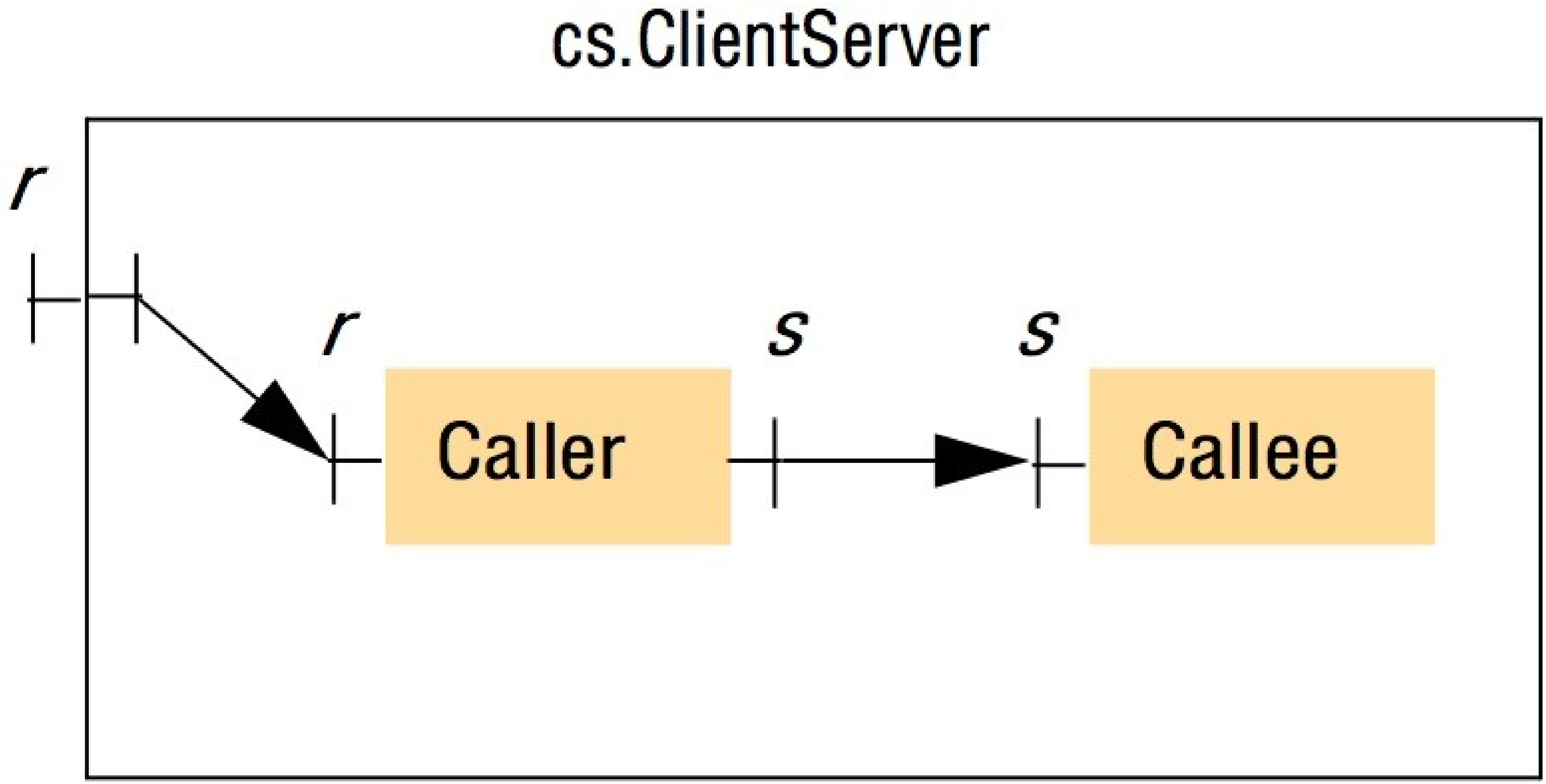
# Figure 8.14
# Transaction attributes in EJB.

| *Attribute* | *Policy* |
|---|---|
| *REQUIRED* | If the client has an associated transaction running, execute within this transaction; otherwise, start a new transaction. |
| *REQUIRES_NEW* | Always start a new transaction for this invocation. |
| *SUPPORTS* | If the client has an associated transaction, execute the method within the context of this transaction; if not, the call proceeds without any transaction support. |
| *NOT_SUPPORTED* | If the client calls the method from within a transaction, then this transaction is suspended before calling the method and resumed afterwards – that is, the invoked method is excluded from the transaction. |
| *MANDATORY* | The associated method must be called from within a client transaction; if not, an exception is thrown. |
| *NEVER* | The associated methods must not be called from within a client transaction; if this is attempted, an exception is thrown. |

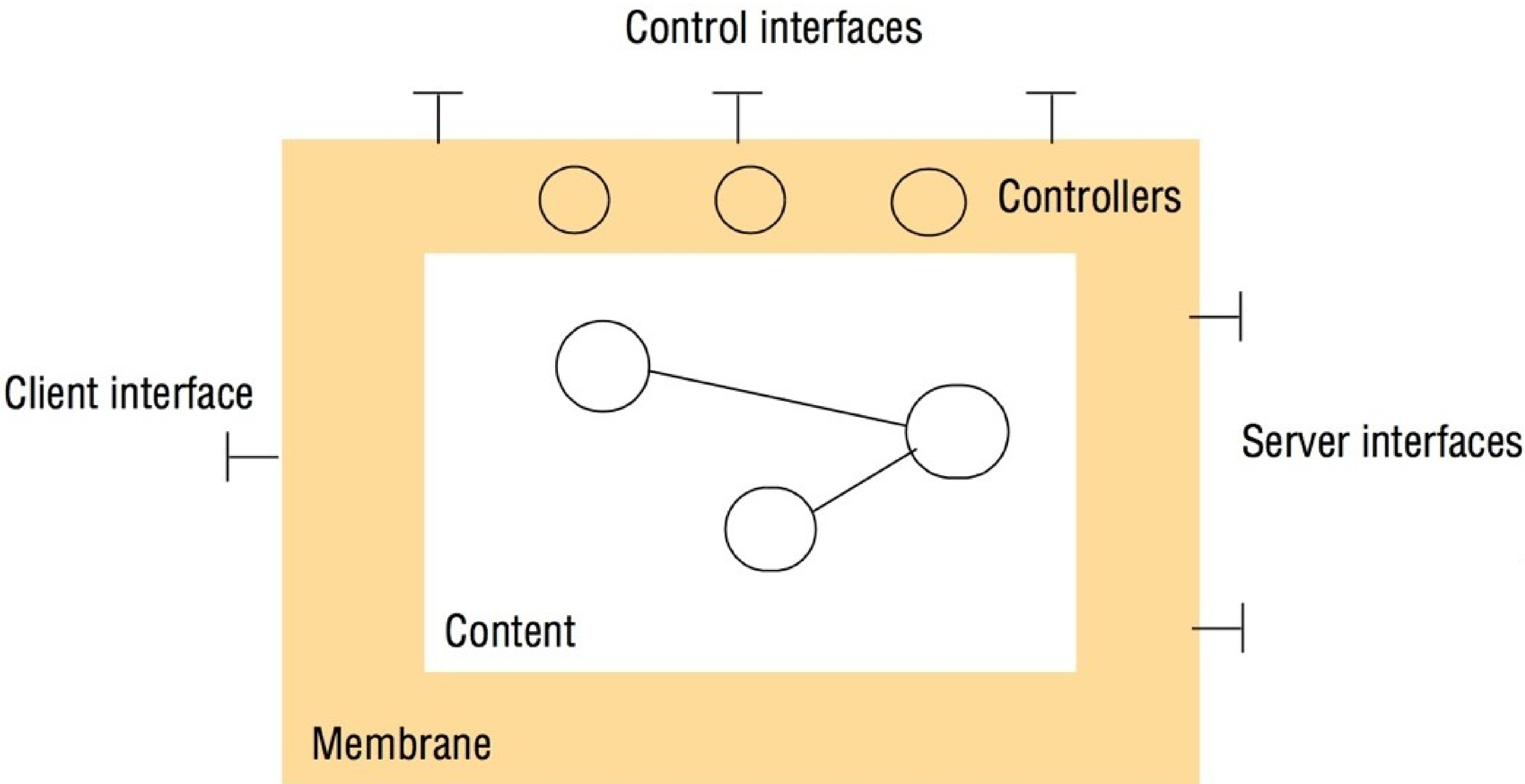
# Figure 8.15
## Invocation contexts in EJB

| *Signature* | *Use* |
|---|---|
| *public Object getTarget()* | Returns the bean instance associated with the incoming invocation or event |
| *public Method getMethod()* | Returns the method being invoked |
| *public Object[] getParameters()* | Returns the set of parameters associated with the intercepted business method |
| *public void setParameters( Object[] params)* | Allows the parameter set to be altered by the interceptor, assuming type correctness is maintained |
| *public Object proceed() throws Exception* | Execution proceeds to next interceptor in the chain (if any) or the method that has been intercepted |

# Figure 8.16
# An example component configuration in Fractal



# Figure 8.17
# The structure of a Fractal component



# Figure 8.18
## *Component* and *ContentController* Interfaces in Fractal

```
public interface Component {
    Object[] getFcInterfaces ();
    Object getFcInterface (String itfName);
    Type getFcType ();
}

public interface ContentController {
    Object[] getFcInternalInterfaces ();
    Object getFcInterfaceInterface(String itfName);
    Component[] getFcSubComponents ();
    void addFcSubComponent (Component c);
    void removeFcSubComponent(Component c);
}
```