Distributed Systems: Concepts and Design

Edition 5

By George Coulouris, Jean Dollimore, Tim Kindberg and Gordon Blair
Addison-Wesley ©Pearson Education 2012

Chapter 8 Exercise Solutions

8.1 The Task Bag is an object that stores (*key*, *value*) pairs. A *key* is a string and a *value* is a sequence of bytes. Its interface provides the following remote methods:

pairOut, with two parameters through which the client specifies a *key* and a *value* to be stored.

pairIn, whose first parameter allows the client to specify the *key* of a pair to be removed from the Task Bag. The *value* in the pair is supplied to the client via a second parameter. If no matching pair is available, an exception is thrown.

readPair, which is the same as *pairIn* except that the pair remains in the Task Bag.

Use CORBA IDL to define the interface of the Task Bag. Define an exception that can be thrown whenever any one of the operations cannot be carried out. Your exception should return an integer indicating the problem number and a string describing the problem. The Task Bag interface should define a single attribute giving the number of tasks in the bag.

8.1 Ans.

Note that sequences must be defined as *typedefs*. *Key* is also a *typedef* for convenience.

```
typedef string Key;
typedef sequence<octet> Value;
interface TaskBag {
    readonly attribute long numberOfTasks;
    exception TaskBagException { long no; string reason; };
    void pairOut (in Key key, in Value value) raises (TaskBagException);
    void pairIn (in Key key, out Value value) raises (TaskBagException);
    void readPair (in Key key, out Value value) raises (TaskBagException);
};
```

8.2 Define an alternative signature for the methods *pairIn* and *readPair*, whose return value indicates when no matching pair is available. The return value should be defined as an enumerated type whose values can be *ok* and *wait*. Discuss the relative merits of the two alternative approaches. Which approach would you use to indicate an error such as a key that contains illegal characters?

8.2 Ans.

```
enum status { ok, wait};
status pairIn (in Key key, out Value value);
status readPair (in Key key, out Value value);
```

It is generally more complex for a programmer to deal with an exception that an ordinary return because exceptions break the normal flow of control. In this example, it is quite normal for the client calling *pairIn* to find that the server hasn't yet got a matching pair. Therefore an exception is not a good solution.

For the key containing illegal characters it is better to use an exception: it is an error (and presumably an unusual occurrence) and in addition, the exception can supply additional information about the error. The client must be supplied with sufficient information to recognise the problem.

8.3 Which of the methods in the Task Bag interface could have been defined as a *oneway* operation? Give a general rule regarding the parameters and exceptions of *oneway* methods. In what way does the meaning of the *oneway* keyword differ from the remainder of IDL?

8.3 Ans.

A *oneway* operation cannot have *out* or *inout* parameters, nor must it raise an exception because there is no reply message. No information can be sent back to the client. This rules out all of the Task Bag operations. The *pairOut* method might be made one way if its exception was not needed, for example if the server could guarantee to store every pair sent to it. However, *oneway* operations are generally implemented with *maybe* semantics, which is unacceptable in this case. Therefore the answer is *none*.

General rule. Return value *void*. No *out* or *inout* parameters, no user-defined exceptions.

The rest of IDL is for defining the interface of a remote object. But *oneway* is used to specify the required quality of delivery.

8.4 CORBA supports passing non-CORBA objects by value. What are the properties of these non-CORBA objects? What are their limitations?

8.4 Ans.

These non-CORBA objects are object-like in the sense that they possess both attributes and methods. However, they are purely local objects in that their operations cannot be invoked remotely. The pass-by-value facility provides the ability to pass a copy of a non-CORBA object between client and server.

8.5 In Figure 8.2 the type *All* was defined as a sequence of fixed length. Redefine this as an array of the same length. Give some recommendations as to the choice between arrays and sequences in an IDL interface.

8.5 Ans.

```
typedef Shape All[100];
```

Recommendations

- if a fixed length structure then use an array.
- if variable length structure then use a sequence
- if you need to embed data within data, then use a sequence because one may be embedded in another
- if your data is sparse over its index, it may be better to use a sequence of <index, value> pairs.

8.6 The Task Bag is intended to be used by cooperating clients, some of which add pairs (describing tasks) and others of which remove them (and carry out the tasks described). When a client is informed that no matching pair is available, it cannot continue with its work until a pair becomes available. Define an appropriate callback interface for use in this situation.

8.6 Ans.

This callback can send the value required by a *readPair* or *pairIn* operation. Its method should not be a *oneway* as the client depends on receiving it to continue its work.

```
interface TaskBagCallback{
    void data(in Value value);
}
```

8.7 What are CORBA security services used for?

8.7 Ans.

They support a range of security mechanisms including authentication, access control, secures communication, auditing and non-repudiation.

8.8 The grammar of IDL is a subset of ANSI C++ with additional constructs to support method signatures. Give the uses of CORBA IDL modules and interfaces.

8.8 Ans.

IDL modules: The module construct allows interfaces and other IDL type definitions to be grouped in logical units. A *module* defines a naming scope, which prevents names defined within a module clashing with names defined outside it.

IDL interfaces: As we have seen, an IDL interface describes the methods that are available in CORBA objects that implement that interface. Clients of a CORBA object may be developed just from the knowledge of its IDL interface.

8.9 Use the Java IDL compiler to process the interface you defined in Exercise 8.1. Inspect the definition of the signatures for the methods *pairIn* and *readPair* in the generated Java equivalent of the IDL interface. Look also at the generated definition of the holder method for the *value* argument for the methods *pairIn* and *readPair*. Now give an example showing how the client will invoke the *pairIn* method, explaining how it will acquire the value returned via the second argument.

8.9 Ans.

The Java interface is:

```
public interface TaskBag extends org.omg.CORBA.Object {
   void pairOut(in Key p, in Value value);
   void pairIn(in Key p, out ValueHolder value);
   void readPair(in Key p, out ValueHolder value);
   int numberOfTasks();
}
```

The class *ValueHolder* has an instance variable

```
public Value value;
```

and a constructor

```
public ValueHolder(Value __arg) {
   value = __arg;
}
```

The client must first get a remote object reference to the TaskBag (see Figure 20.5). probably via the naming service.

```
...
TaskBag taskBagRef = TaskBagHelper.narrow(taskBagRef.resolve(path));
Value aValue;
taskbagRef.pairIn("Some key", new ValueHolder(aValue));
```

The required value will be in the variable *aValue*.

8.10 What are the new variants to the invocation semantics used by asynchronous RMI?

8.10 Ans.

Asynchronous RMI adds two new variants to the invocation semantics of RMIs:

- *callback*, in which a client uses an extra parameter to pass a reference to a callback with each invocation, so that the server can call back with the results;
- and *polling*, in which the server returns a *valuetype* object that can be used to poll or wait for the reply.

8.11 "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only." What is the significance of the word 'only' in this statement?

8.11 Ans.

The phrase "only" refers to the fact that any context dependencies must be explicit, that is there are no implicit dependencies present.

8.12 Containers support a common pattern often encountered in distributed applications. What does this pattern consist of?

8.12 Ans.

- A front-end (perhaps web-based) client.
- A container holding one or more components which implement the application or business logic.
- System services that manage the associated data in persistent storage.

8.13 What are the key roles of the EJB specification? Briefly explain each.

8.13 Ans.

- The *bean provider* who develops the application logic of the component(s);
- The *application assembler* who assembles beans into application configurations;
- The *deployer* who takes a given application assembly and ensures it can be correctly deployed in a given operational environment;
- The *service provider* is a specialist in fundamental distributed system services such as transaction management and establishes the desired level of support in these areas;
- The *persistence provider* is a specialist in mapping persistent data to underlying databases and in managing this relationship at runtime;
- The *container provider* builds on the above two roles and is responsible for correctly configuring containers with the required level of distributed systems support in terms of non-functional properties related to, for example, transactions and security as well as the desired support for persistence;
- The *system administrator* is responsible for monitoring a deployment at runtime and making any adjustments to ensure its correct operation.

8.14 What are the transaction attributes in EJB?

8.14 Ans.

REQUIRED, REQUIRES_NEW, SUPPORTS, NOT_SUPPORTED, MANDATORY, NEVER.

8.15 Explain carefully how component-based middleware in general and EJB in particular can overcome the key limitations of distributed object middleware. Provide examples to illustrate your answer.

8.15 Ans.

The problems and associated solutions are described below.

Implicit dependencies: Component-based middleware get round this issue by having a complete contract with the environment (and other components) through both required and provided interfaces. Required interfaces in particular act as declarations of dependencies on other components that need to be there and bound to this component for it to function correctly. This eliminates implicit dependencies. In EJB, this is achieved through the mechanism of dependency injection. As an example, consider making calls to a transaction manager. In distributed object middleware such calls may well be embedded in the code of an object and this is not visible from outside the encapsulation. In EJB, this would be declared as a dependency, as in the call the following call taken from Section 8.5.1:

@Resource javax.transaction.UserTransaction ut;

Programming complexity: ?Programming complexity is concerned with the need to make low level calls to the underlying middleware. This is particularly true in CORBA where the Portable Object Adaptor (POA) and ORB Core, for example, offers a relatively sophisticated interfaces for areas such as the creation and management of object references, the management of object lifecycles, activation and passivation policies, the management of persistent state and policies for mappings to underlying platform resources such as threads. Components hide these interfaces by doing such management within the container abstraction. EJB does not have to deal with the complexities of CORBA but still significantly simplifies lifecycle management, for example.

Lack of separation of distribution concerns: There is additional complexity in managing calls to distributed systems services such as security or transaction services. Again, this is hidden from the programmer through the container abstraction. In EJB, this is particularly easy to deal with through a combination of configuration by exception and the use of annotations to declare requirements more declaratively. An example is providing support for transaction management which, in the simplest case (container-managed transactions), can be set up with simple annotations such as:

@TransactionManagement (CONTAINER)

No support for deployment: All component technologies provide support for deployment through the packaging of components with associated architectural descriptions and deployment descriptors. Tools are provided to interpret such packaging and deploy the associated configurations. In EJB, jar files fulfil this purpose.

8.16 Discuss whether the EJB architecture would be suitable to implement a massively multiplayer online game (an application domain initially introduced in Section 1.2.2). What would be the strengths and weaknesses of using EJB in this domain?

8.16 Ans.

The EJB architecture makes it easier to develop distributed applications by hiding much of the complexity from the programmer. It does this by imposing a particular architectural pattern on applications, that is one where a potentially large number of clients access fairly coarse granularity resources and this needs protection in terms of security and transaction management, a pattern which works best for applications such as eCommerce or banking applications that fit naturally this style of architecture.

Massively multiplayer online games are quite a different class of application. For example, they require low latency interaction with the world views and also may tolerate weak consistency to support such low latency interaction. It is likely that the container-style interaction would not be right for this application domain although it would provide some support for the development of such applications. More importantly, it is likely that the non-functional properties (and the associated policies) provided for containers would not be right for this domain, for example for transaction management (multiplayer online games would not require full transaction management). There would also be question marks over whether an EJB implementation would scale in the right way to large number of online players, all requiring low latency interactions.

8.17 What are the main methods associated with the invocation context?

8.17 Ans.

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.

8.18 Fractal supports *bindings* between interfaces. What are the two styles of binding supported by the model?

8.18 Ans.

Primitive bindings: The simplest style of binding is a primitive binding which is a direct mapping between one client interface and one server interface within the same address space, assuming the types are compatible.

Composite bindings: Fractal also supports composite bindings which are arbitrarily complex software architectures (that is consisting of components and bindings) implementing communication between a number of interfaces potentially on different machines.

8.19 What is the use of the Fractal interface OpenCOM?

8.19 Ans.

OpenCOM is a lightweight component model with very similar goals to Fractal. In particular, OpenCOM is a minimal and open component model that is designed to be domain- and operating environment-independent, that is the component technology is sufficiently flexible to be applied in any context including demanding areas including resource-limited wireless sensor networks.

8.20 Fractal defines a programming model and, as such, is programming language agnostic. Name the languages in which the implementations of this model are available.

8.20 Ans.

- Julia and AOKell (Java-based with the latter also offering support for aspect-oriented programming);
- Cecilia and Think (C-based);
- FracNet (.Net based);
- FracTalk (Smalltalk-based);
- Julio (Python-based).

