8 Distributed objects and components

8.1 Introduction

Distributed object middleware

- *encapsulation* in object-based solutions – well suited to distributed programming
- *data abstraction* – clean separation between the specification of an object and its implementation ⇒ programmers to deal solely in terms of interfaces and not concern with implementation details
- ⇒ more dynamic and extensible solutions

Examples of distributed objects middleware: Java RMI and CORBA

Component-based middleware

– to overcome a number of limitations with distributed object middleware:

Implicit dependencies: Object interfaces do not describe what the implementation of an object depends on

Programming complexity: need to master many low-level details

Lack of separation of distribution concerns: Application developers need to consider details of security, failure handling and concurrency – largely similar from one application to another

No support for deployment: Object-based middleware provides little or no support for the deployment of (potentially complex) configurations of objects

8.2 Distributed objects

- DS started as client-server architecture
- with emergence of highly popular OO languages (C++, Java) the OO concept spreading to DS
- Unified Modelling Language (UML) in SE has its role too in middleware developments (e.g. CORBA and UML standards developed by the same organisation)

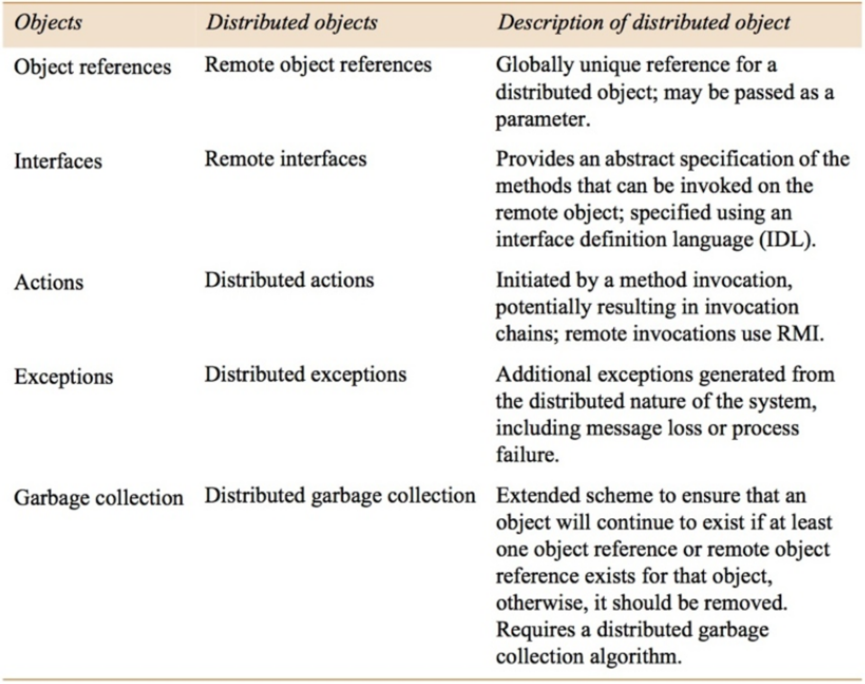
Distributed object (DO) middleware

- Java RMI and CORBA – quite common
- but CORBA – language independent

in DO he term class is avoided – instead factory instantiating new objects from a given template

- in Smalltalk – implementational inheritance
- in DO – interface inheritance:
  - new interface inherits the method signatures of the original interface
    - + can add extra ones

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.

OO: objects + class + inheritance $\longleftrightarrow$ DO: encapsulation + data abstraction + design methodologies

The added complexities with DO:

- *Inter-object communication*
  - remote method invocation
  - + often other communications paradigms
    - (e.g. CORBA's event service + associated notification service)
- *Lifecycle management*
  - creation, migration and deletion of DO

- *Activation and deactivation*
  - # DOs may be very large...
  - node availabilities
- *Persistence*

  state of DO need to be preserved across all cycles (like [de]activation, system failures etc.)
- *Additional services*
  - e.g. naming, security and transaction services