Lecture 15 – Distributed Object Architectures (Sommerville Ch. 11)

Karl R. Wilcox
K.R.Wilcox@reading.ac.uk

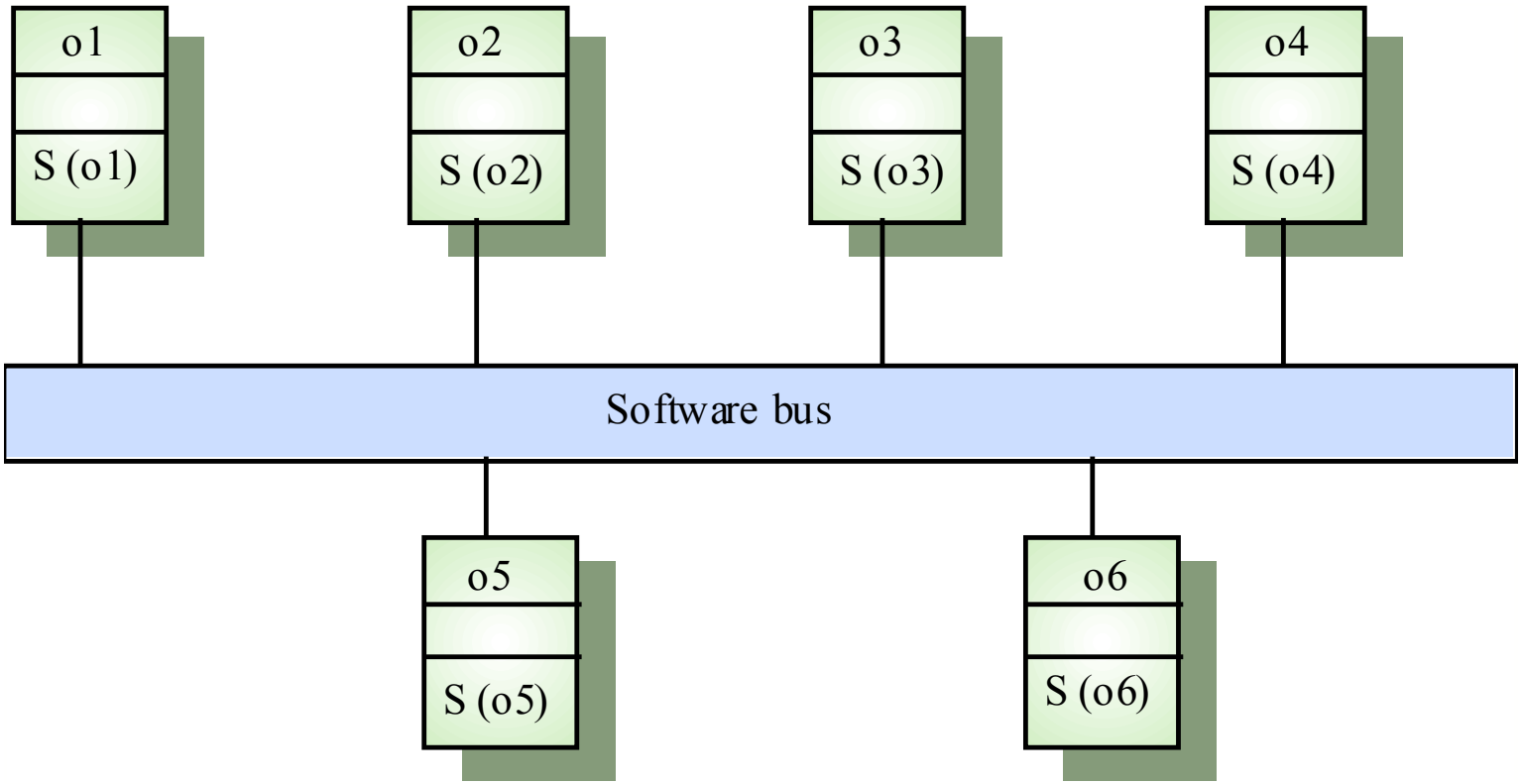
Objectives

- To explain the differences between client-server and distributed object architectures
- To describe object request brokers and the principles underlying the CORBA standards

Distributed object architectures

- There is no distinction in a distributed object architectures between clients and servers
- Each distributable entity is an object that provides services to other objects and receives services from other objects
- Object communication is through a middleware system called an object request broker (software bus)
- However, more complex to design than C/S systems

Distributed object architecture



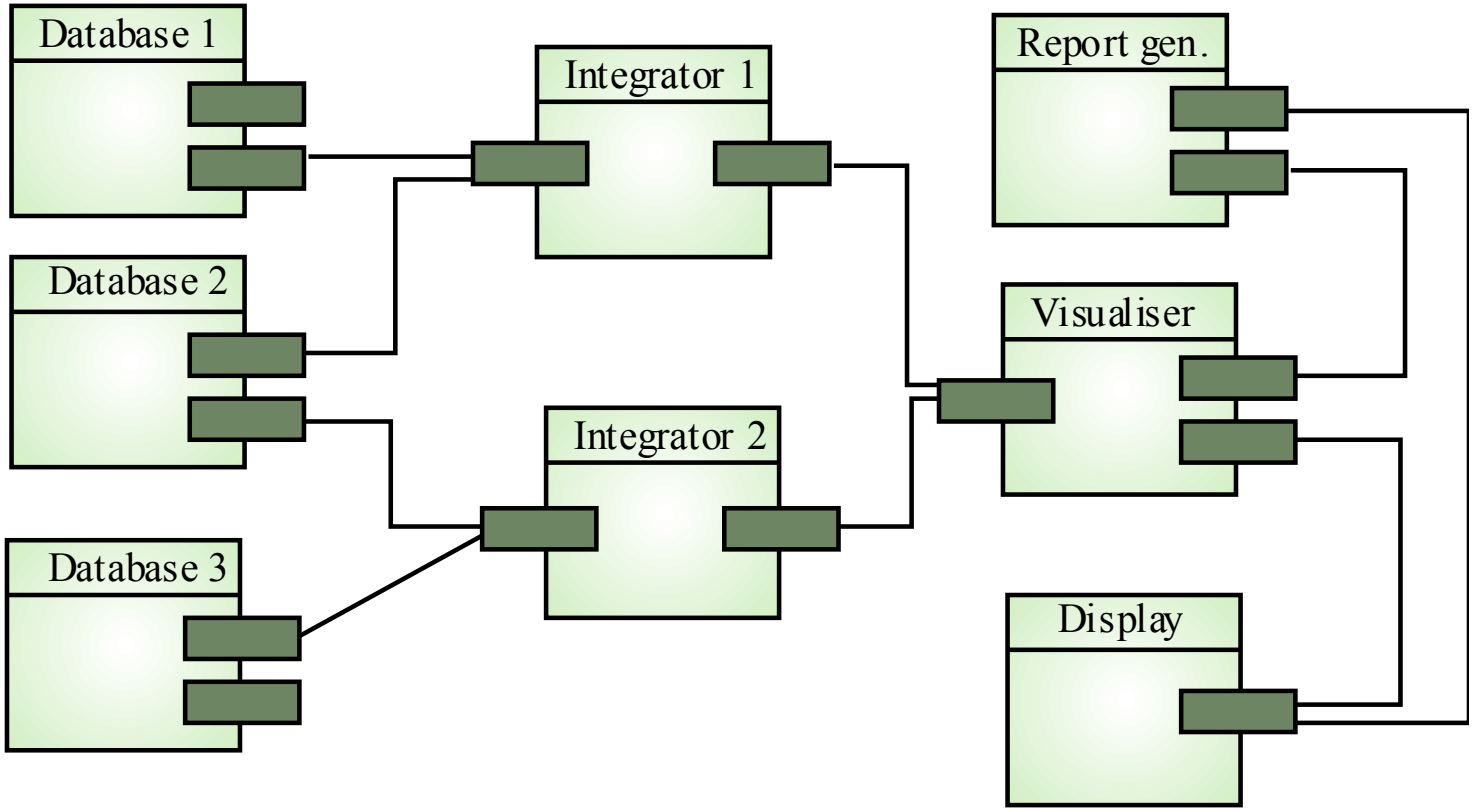
Advantages of distributed object architecture

- It allows the system designer to delay decisions on where and how services should be provided
- It is a very open system architecture that allows new resources to be added to it as required
- The system is flexible and scaleable
- It is possible to reconfigure the system dynamically with objects migrating across the network as required

Uses of distributed object architecture

- As a logical model that allows you to structure and organise the system. In this case, you think about how to provide application functionality solely in terms of services and combinations of services
- As a flexible approach to the implementation of client-server systems. The logical model of the system is a client-server model but both clients and servers are realised as distributed objects communicating through a software bus

A data mining system



Data mining system

- The logical model of the system is not one of service provision where there are distinguished data management services
- It allows the number of databases that are accessed to be increased without disrupting the system
- It allows new types of relationship to be mined by adding new integrator objects

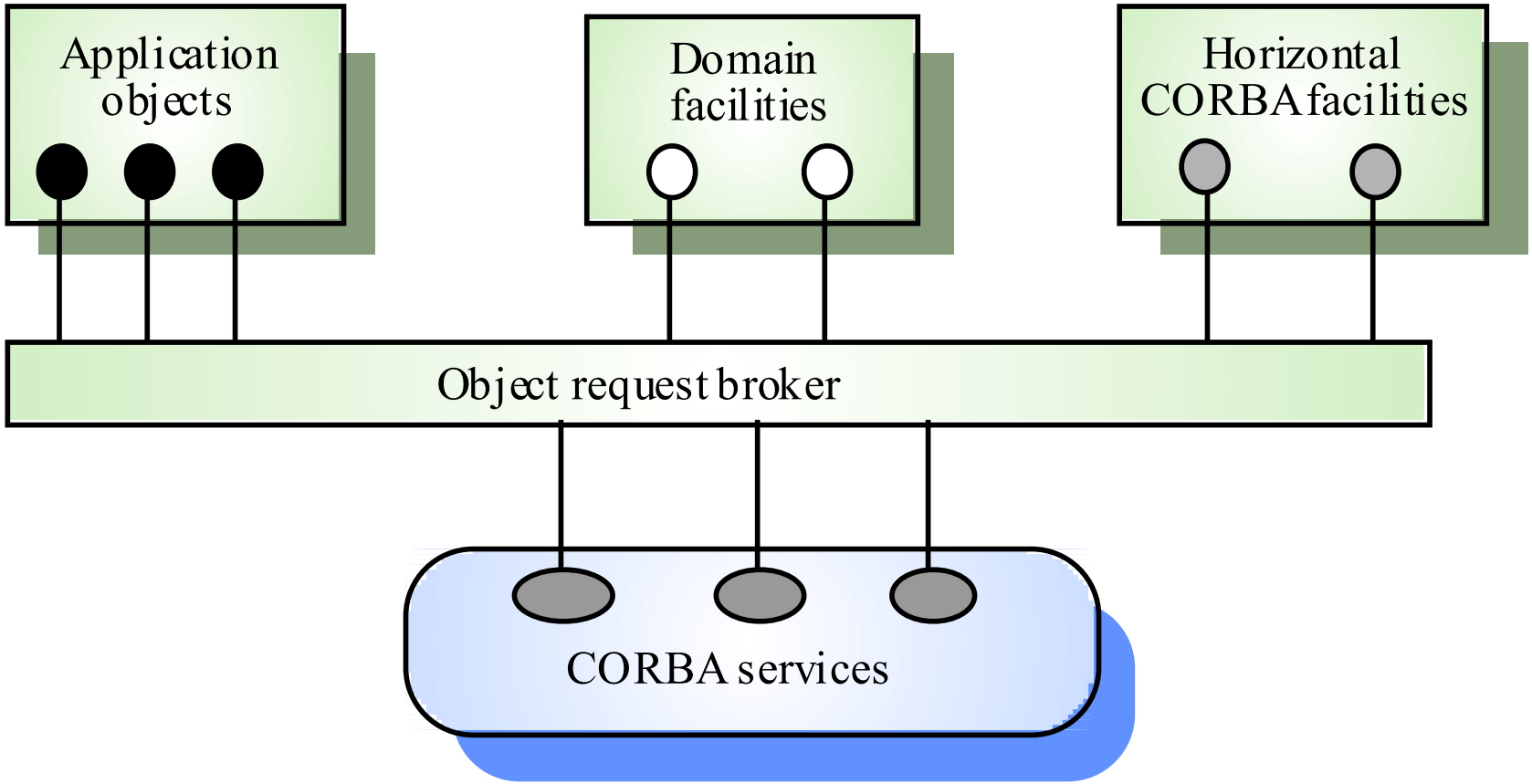
CORBA

- CORBA is an international standard for an Object Request Broker - middleware to manage communications between distributed objects
- Several implementation of CORBA are available
- DCOM is an alternative approach by Microsoft to object request brokers
- CORBA has been defined by the Object Management Group

Application structure

- Application objects
- Standard objects, defined by the OMG, for a specific domain e.g. insurance
- Fundamental CORBA services such as directories and security management
- Horizontal (i.e. cutting across applications) facilities such as user interface facilities

CORBA application structure



CORBA standards

- An object model for application objects
  - A CORBA object is an encapsulation of state with a well-defined, language-neutral interface defined in an IDL (interface definition language)
- An object request broker that manages requests for object services
- A set of general object services of use to many distributed applications
- A set of common components built on top of these services

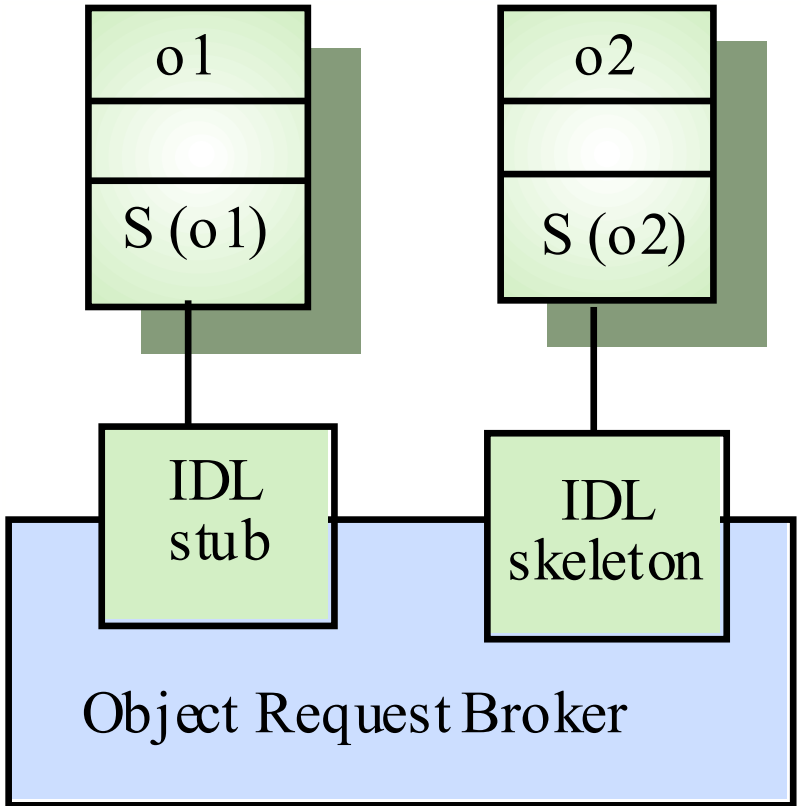
CORBA objects

- CORBA objects are comparable, in principle, to objects in C++ and Java
- They MUST have a separate interface definition that is expressed using a common language (IDL) similar to C++
- There is a mapping from this IDL to programming languages (C++, Java, etc.)
- Therefore, objects written in different languages can communicate with each other

Object request broker (ORB)

- The ORB handles object communications. It knows of all objects in the system and their interfaces
- Using an ORB, the calling object binds an IDL stub that defines the interface of the called object
- Calling this stub results in calls to the ORB which then calls the required object through a published IDL skeleton that links the interface to the service implementation

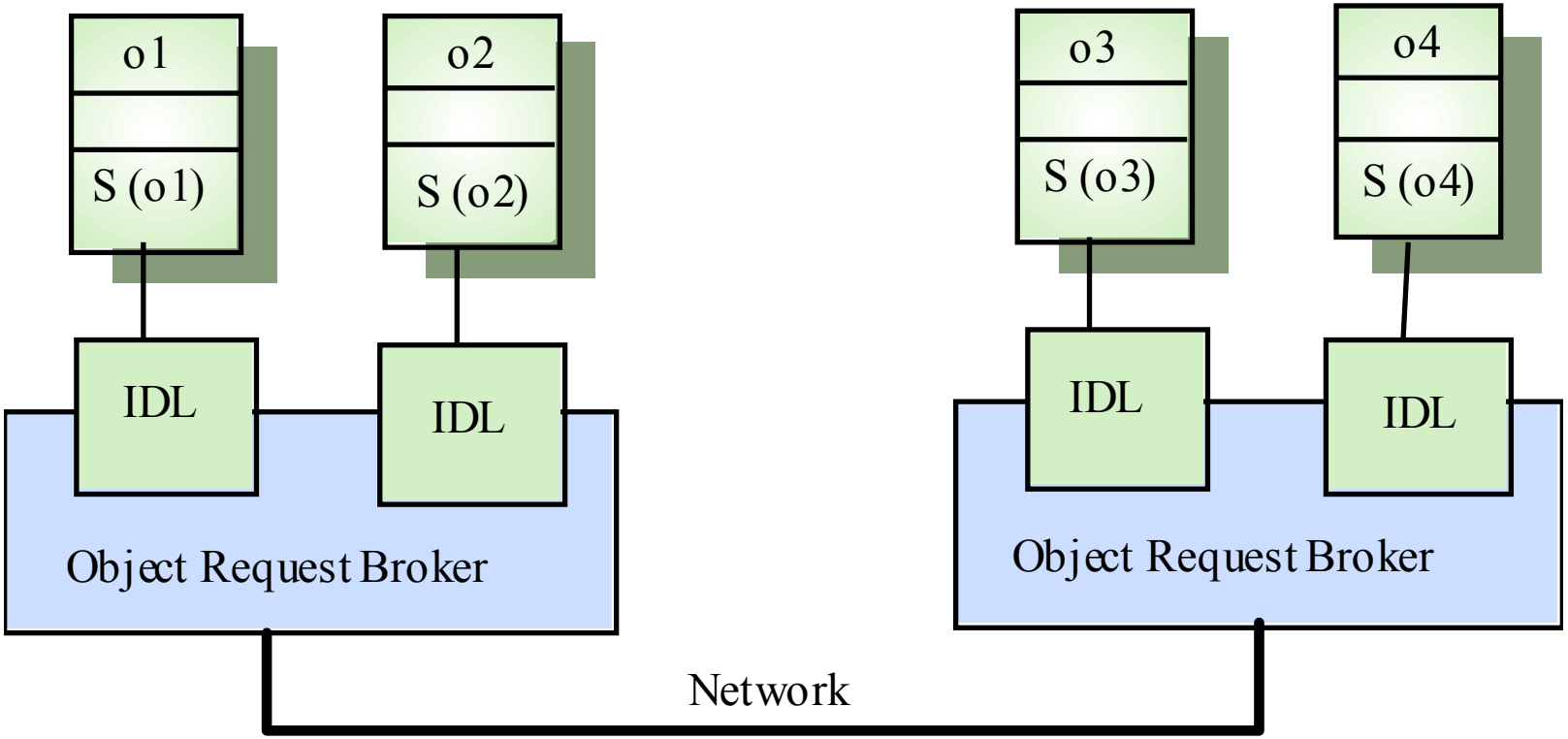
ORB-based object communications



Inter-ORB communications

- ORBs are not usually separate programs but are a set of objects in a library that are linked with an application when it is developed
- ORBs handle communications between objects executing on the sane machine
- Several ORBS may be available and each computer in a distributed system will have its own ORB
- Inter-ORB communications are used for distributed object calls

Inter-ORB communications



CORBA services

- **Naming and trading services**
  - **These allow objects to discover and refer to other objects on the network**
- **Notification services**
  - **These allow objects to notify other objects that an event has occurred**
- **Transaction services**
  - **These support atomic transactions and rollback on failure**

Some Real ORB Findings - 1

- The basic ORB provided no security – once a connection had been made to the Trader service then it would freely publish all of the fixed objects and methods it trades. (This could be mitigated by only having one fixed object which would create dynamic objects as required, passing the object reference back over the fixed event channel – this would involve extra development however).
- The ORB did not implement a naming service – the client had to know the object reference of the trader service.
- Built in resilience was minimal – it was possible to specify a master and secondary trader however there was no indication which service had responded, and hence whether the primary service had failed.

Some Real ORB Findings - 2

- The ORB (at least in the version we had) did not any systems management capability (SNMP or WBEM). Failure of the ORB daemon resulted in a “failed connection” indication being returned by the IIOP call but did not provide any details.
- Buffering on the event channel was not very flexible. We could set the level of buffering to whatever value we wanted but the methods supported where “give me the latest event” or “give me every event in your buffer”.

Key points

- In a distributed object architecture, there is no distinction between clients and servers
- Distributed object systems require middleware to handle object communications
- The CORBA standards are a set of middleware standards that support distributed object architectures