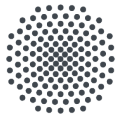


Universität Stuttgart
Master of Science

Objects and Components

Marco Aiello

Distributed Object Middleware

- Encapsulation
- Data Abstraction
- Dynamic and extensible

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.

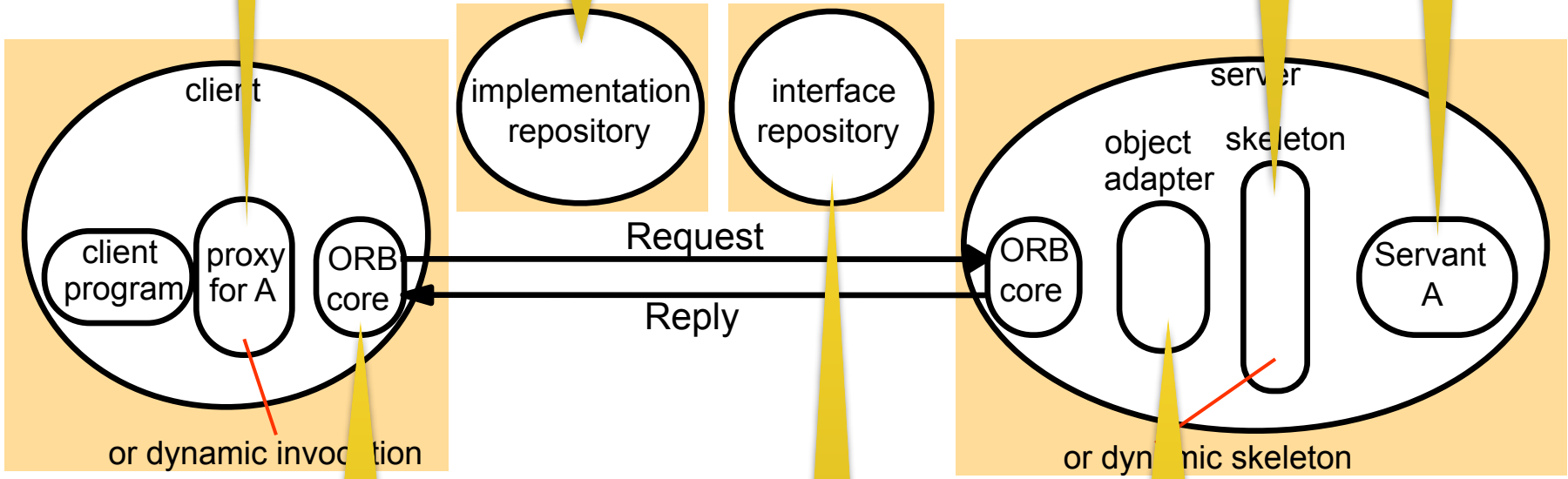
The main components of the CORBA architecture

- created in the language of the client by IDL compiler
- marshall/unmarshall

- implement objects
- created on demand (e.g. db)

- activate or locate registered servers on demand
- mapping names to pathnames of files with object code
- callback objects don't need it (run only once)
- access rights and redundancy are possible

- standardized for portability
- may create separate threads
- created in the language of the server by IDL compiler



- start/stop
- convert reference to string
- argument list for dynamic invocation

- information about registered interfaces
- helpful for reflection

- create remote object references
- dispatch via skeleton
- activate/deactivate servants
- gives objects unique name

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]

From Objects to Components

- implicit dependencies
  - internal object behavior hidden (e.g., cascading RMIs)
  - specify not only interface, but also the dependencies of an object
- programming complexity
  - separate code for distributed application from middleware operation
- lack of separation of distribution concerns
  - security, transactions, coordination, replication
- no support for deployment
  - transparent deployment of a distributed application as a whole rather than as many one machine deployments

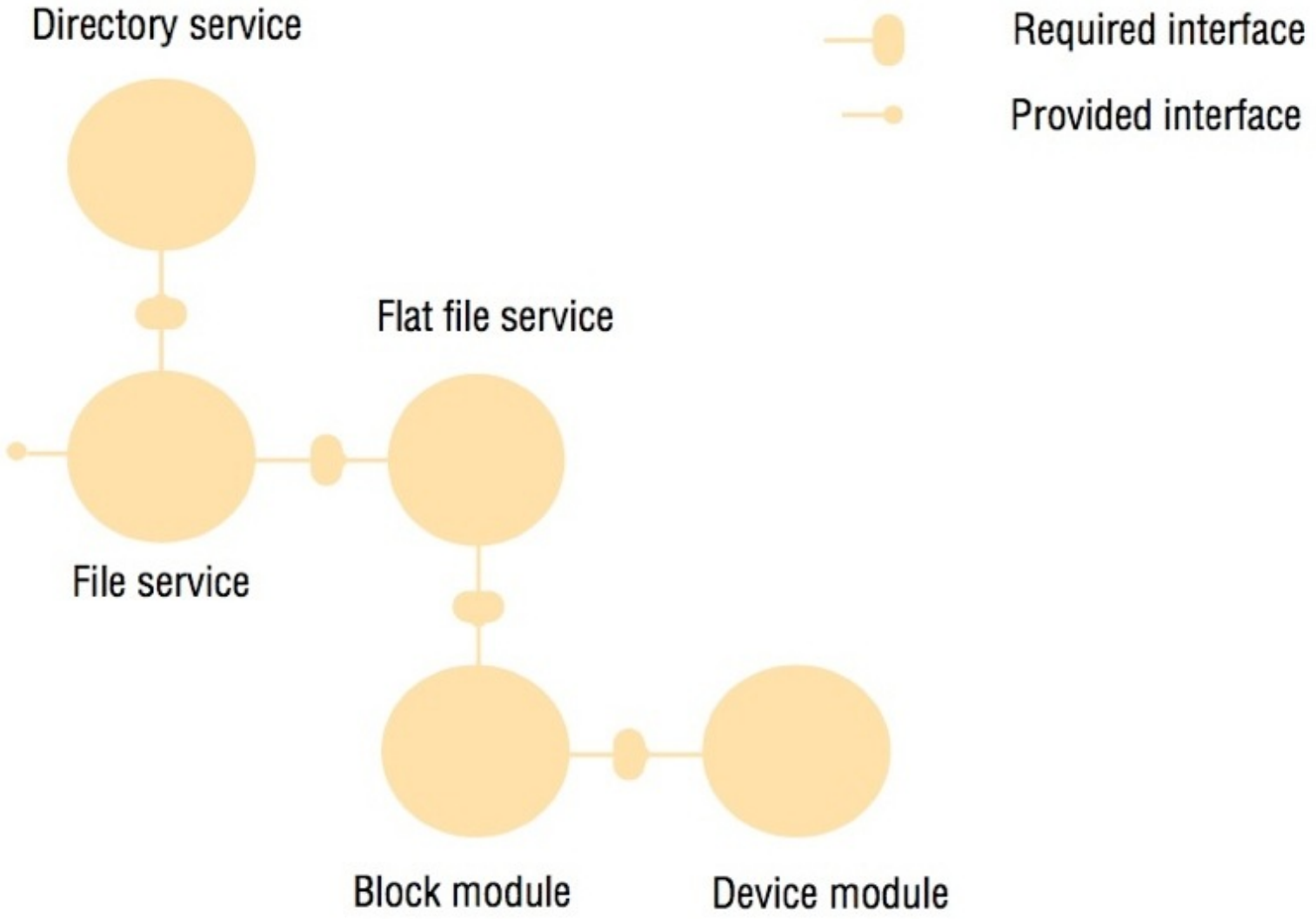
Components

- "*A software component is a unit of composition with contractually specified interfaces and context dependencies only*" [Szyperski, 2002]

- Contract includes provided and required interfaces of other components

- Software Architecture: components, interfaces and connection between them.

Figure 8.11
An example software architecture



Components

- Interface supporting remote method invocation or distributed events
- Composition central part of design and programming components
- Containers where deployed components live

- Popular examples: Web service composition / Microservices / Kubernetes

Figure 8.12
The structure of a container