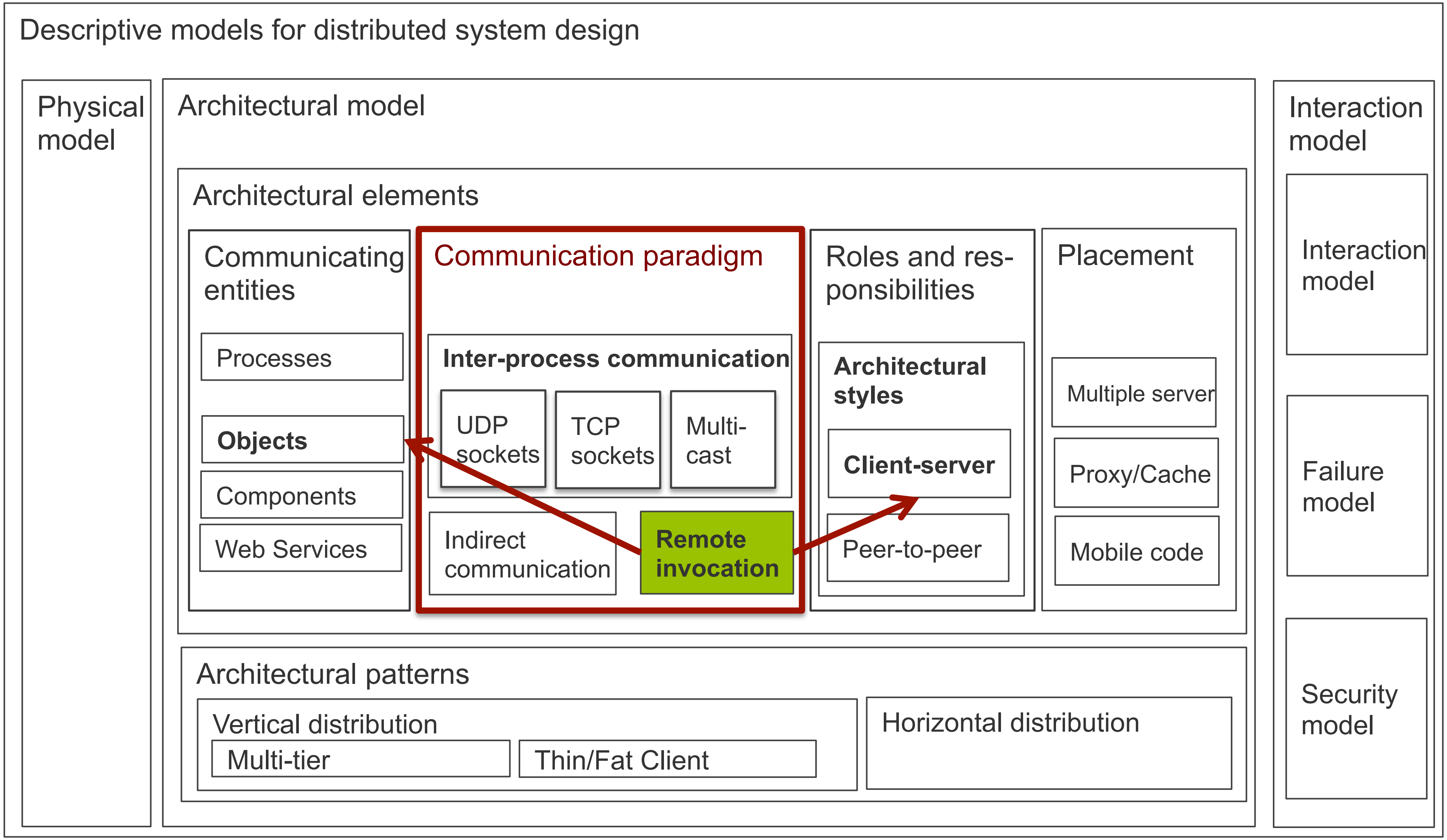
Prof. Dr. Claudia Müller-Birn
Institute for Computer Science, Networked Information Systems

Freie Universität Berlin

Distributed objects and components

Netzprogrammierung
(Algorithmen und Programmierung V)

Our topics last week



Our topics today

- Distributed objects - emergence, key features, examples
- Another example for distributed objects CORBA 2.0 (besides Java RMI)
  - Idea behind CORBA and why it failed
- Problems with object-oriented middleware
- Development of distributed components
  - Idea behind components
- Example for distributed components: Enterprise JavaBeans

Distributed objects and components
Introduction

Benefits of distributed object middleware

The encapsulation inherent in object-based solutions is well suited to distributed programming.

The related property of data abstraction provides a clean separation between the specification of an object and its implementation, allowing programmers to deal solely in terms of interfaces and not be concerned with implementation details such as programming language and operating system used.

This approach also lends itself to more dynamic and extensible solutions, for example by enabling the introduction of new objects or the replacement of one object with another (compatible) object.

Distributed objects and components

Key features of distributed objects

The natural evolution of distributed objects

In distributed systems, earlier middleware was based on the client-server model and there was a desire for more sophisticated programming abstractions.

In programming languages, earlier work in object-oriented languages such as Smalltalk led to the emergence of more mainstream and heavily used programming languages such as Java and C++ (languages used extensively in distributed systems).

In software engineering, significant progress was made in the development of object-oriented design methods, leading to the emergence of the Unified Modeling Language (UML) as an industrial-standard notation for specifying (potentially distributed) object-oriented software systems.

Distributed object middleware

It provides a programming abstraction based on the object-oriented principles.

Leading examples: Java RMI and CORBA

Java RMI

Restricted to the
Java-based development

CORBA

Multi-language solution
allowing objects
written in a variety of languages to interoperate
(bindings exist for example for C++, Java, Python)

Freie Universität Berlin

Distributed objects and components

CORBA 2.0

OMG and CORBA

The Object Management Group (OMG) was formed in 1989 to develop, adopt, and promote standards for the development and deployment of applications in distributed heterogeneous environments. Todays it is focused on modeling (programs, systems and business processes) and model-based standards.

The OMG defined a *Object Management Architecture* (OMA) with one of its key components - the *Common Object Request Broker Architecture* (CORBA) specification.

Object Request Broker (ORB) helps a client to invoke a method on an object.

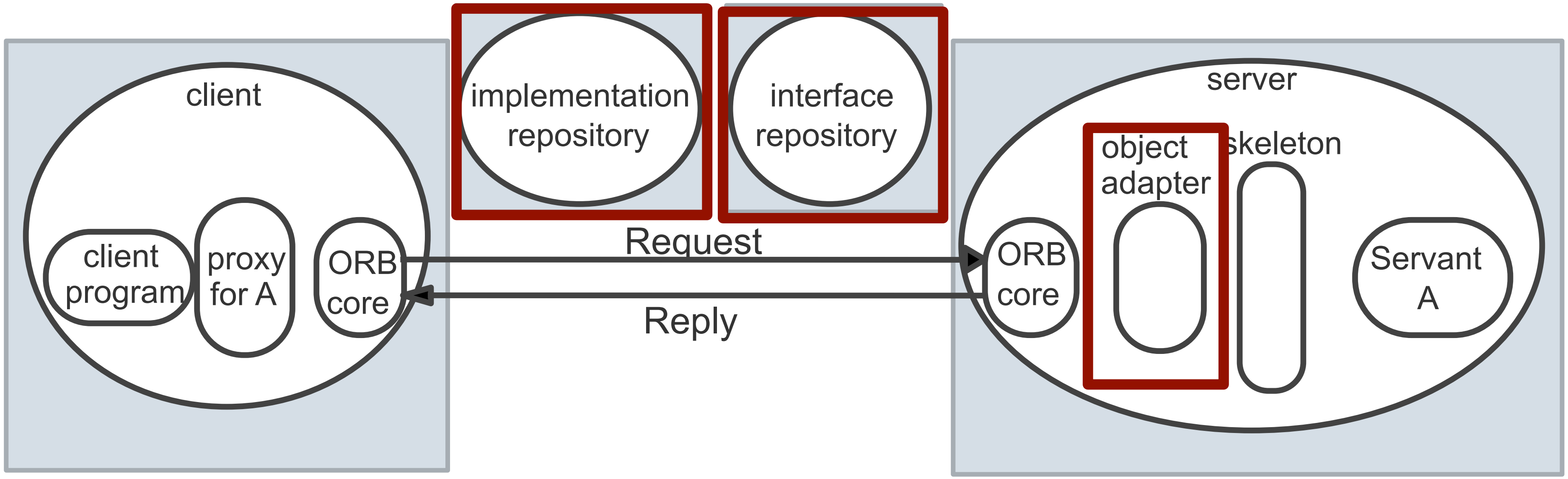
Main components of CORBA 2.0

- An interface definition language known as IDL
- An architecture
- An external data representation, called CDR
- A standard form for remote object references

CORBA 2.0

CORBA’s architecture

Main components of the CORBA architecture



The object request broker (ORB)

All requests are managed by the ORB. This means that every invocation (whether it is local or remote) of a CORBA object is passed to an ORB.

In the case of a remote invocation the invocation passed from the ORB of the client to the ORB of the object implementation

The ORB is responsible for all the mechanisms required to perform these tasks:

- Find the object implementation for the request.
- Prepare the object implementation to receive the request.
- Communicate the data making up the request.

Object adapter

The role of the object adapter is to bridge the gap between CORBA objects with IDL interfaces and the programming language interfaces of the corresponding servant classes.

An object adapter has the following tasks

- It creates remote object references for CORBA objects
- It dispatches each RMI via skeleton to the appropriate servant
- It activates and deactivates servants

Implementation and interface repository

Implementation repository

- Is responsible for activating registered servers on demand and for locating servers that are currently running.
- It stores a mapping from the names of object adapters to the pathnames of files containing object implementations.

Interface repository

- Provides information about registered IDL interfaces to clients and servers that require it. Interfaces can be added to the interface repository service.
- Using the IR, a client should be able to locate an object that is unknown at compile time, find information about its interface, then build a request to be forwarded through the ORB.

CORBA 2.0

CORBA IDL

CORBA's object model

CORBA's object model is very similar to the already known remote method invocation.

In CORBA clients must not necessarily be objects but can be any program that sends request messages to remote objects and receives replies.

CORBA objects refer to remote objects and such a object implements an IDL interface, has a remote object reference and it is able to respond to invocations of methods in its IDL interface.

IDL interfaces Shape and ShapeList

```
struct Rectangle{
    long width;
    long height;
    long x;
    long y;
} ;

struct GraphicalObject {
    string type;
    Rectangle enclosing;
    boolean isFilled;
};

interface Shape {
    long getVersion() ;
    GraphicalObject getAllState() ;          // returns state of the GraphicalObject
};

typedef sequence <Shape, 100> All;
interface ShapeList {
    exception FullException{ };
    Shape newShape(in GraphicalObject g) raises (FullException);
    All allShapes();                         // returns sequence of remote object references
    long getVersion() ;
};
```

Interface Definition Language (IDL)

Specification language

Language independent interface

Declare interfaces to object methods

IDL maps to many high-level programming languages

Design paradigm

Code to interface specified in the IDL regardless of implementation

Many OMG standard mappings, such as C, C++, Java, Python, Smalltalk, Ada

CORBA 2.0

CORBA today

Today, CORBA is used mostly to wire together components that run inside companies' networks, where communication is protected from the outside world by a firewall.

It is also used for real-time and embedded systems development, a sector in which CORBA is actually growing.

Overall, however, CORBA's use is in decline and it cannot be called anything but a niche technology now.

(Henning, 2008)

?

(Henning, 2008)

Some but by far not all reasons

Technical issues

- Many of CORBA's APIs are far larger than necessary. For example, CORBA's object adapter requires more than 200 lines of code interface definitions, even though the same functionality can be provided in 30 lines
- CORBA's unencrypted traffic conflicts with the reality of corporate security policies.

Procedural issues

- There are no entry of qualifications to participate in the standardization process.
- RFPs often call for a technology that is unproven.
- Vendors respond to RFPs even when they have known technical flaws.
- Vendors have a conflict of interest when it comes to standardization.

Distributed objects and components

Distributed components

Issues with object-oriented middleware

Implicit dependencies

- Internal (encapsulated) behavior of the object is hidden, e.g. an object may communicate with other objects or may use other services

➢ Not only a clear interface definition is needed but also the dependencies the object has on other objects in the distributed configuration.

Interaction with the middleware

- Programmers is exposed to many relatively low-level details associated with the middleware architecture - need to further simplifications

➢ Clean separation of concern is needed between code related to operation in a middleware framework and code associated with the application.

Issues with object-oriented middleware (*cont.*)

Lack of separation of distribution concerns

- Programmers have to explicitly deal with non-functional concerns related to issues such as security, coordination, and replication

➢ The complexities of dealing with the distributed system services should be hidden wherever possible from the programmer.

No support for deployment

- Technologies such as Java RMI and CORBA does not support for the deployment of the developed arbitrary distributed configurations

➢ Middleware platforms should provide intrinsic support for deployment so that distributed software can be installed and deployed in the same way as software for a single machine.

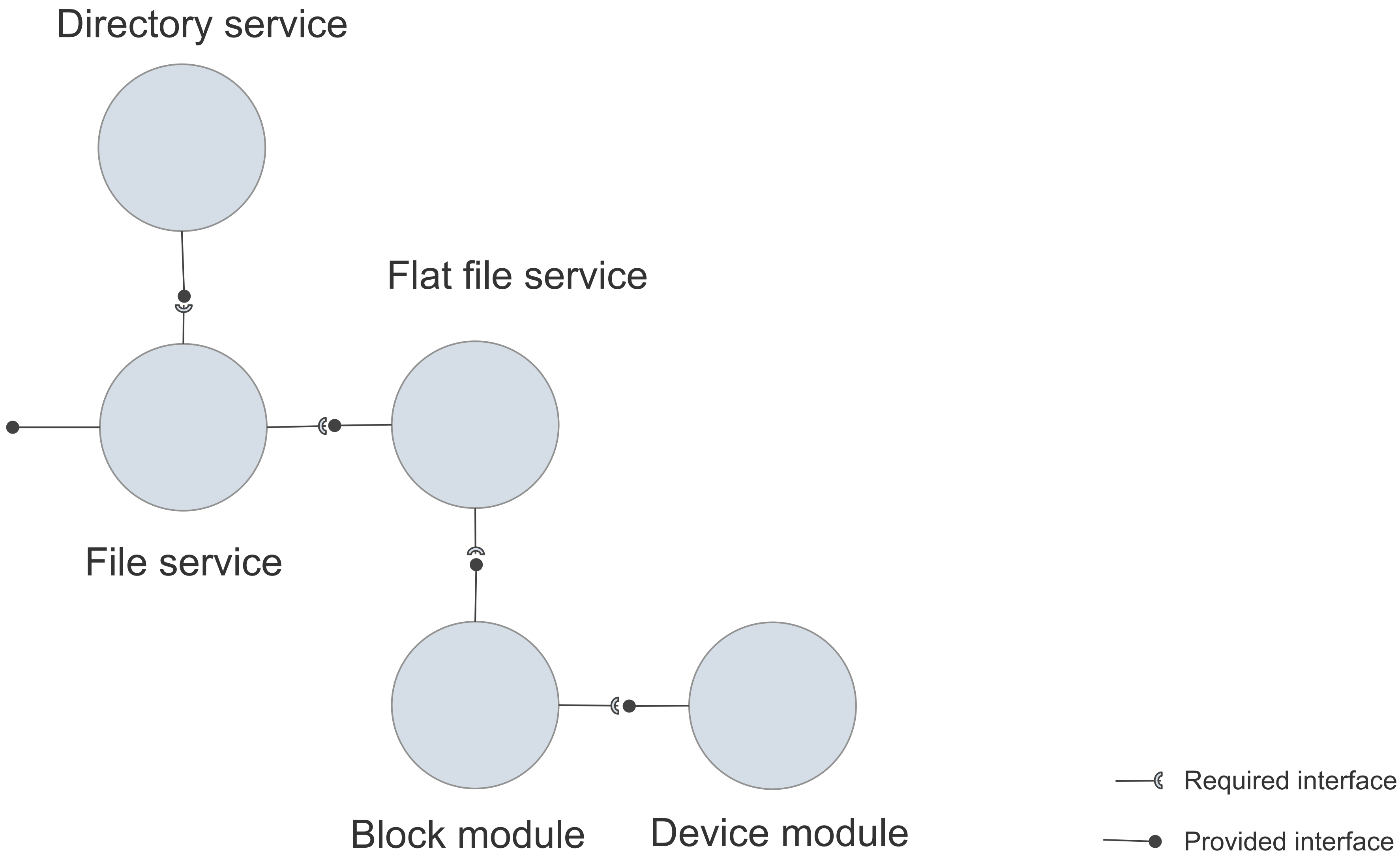
Essence of components

A software component is a unit of composition with contractually specified interfaces and explicit content decencies only.

A component is specified in terms of a contract, which includes

- A set of provided interfaces – that is, interfaces that the component offers as services to other components
- A set of required interfaces – that is, the dependencies that this component has in terms of other components that must be present and connected to this components for it to function correctly.

Example software architecture of a simple file system



What is component-based development?

Programming in component-based systems is concerned with the development of components and their composition.

Goal

- Support a style of software development that parallels hardware development in using off-the-shelf components and composing them together to develop more sophisticated services

It supports third-party development of software components and also make it easier to adapt system configurations at runtime, by replacing one component with another.

Distributed objects and components

Components and distributed systems

Containers

Containers support a common pattern often encountered in distributed systems development

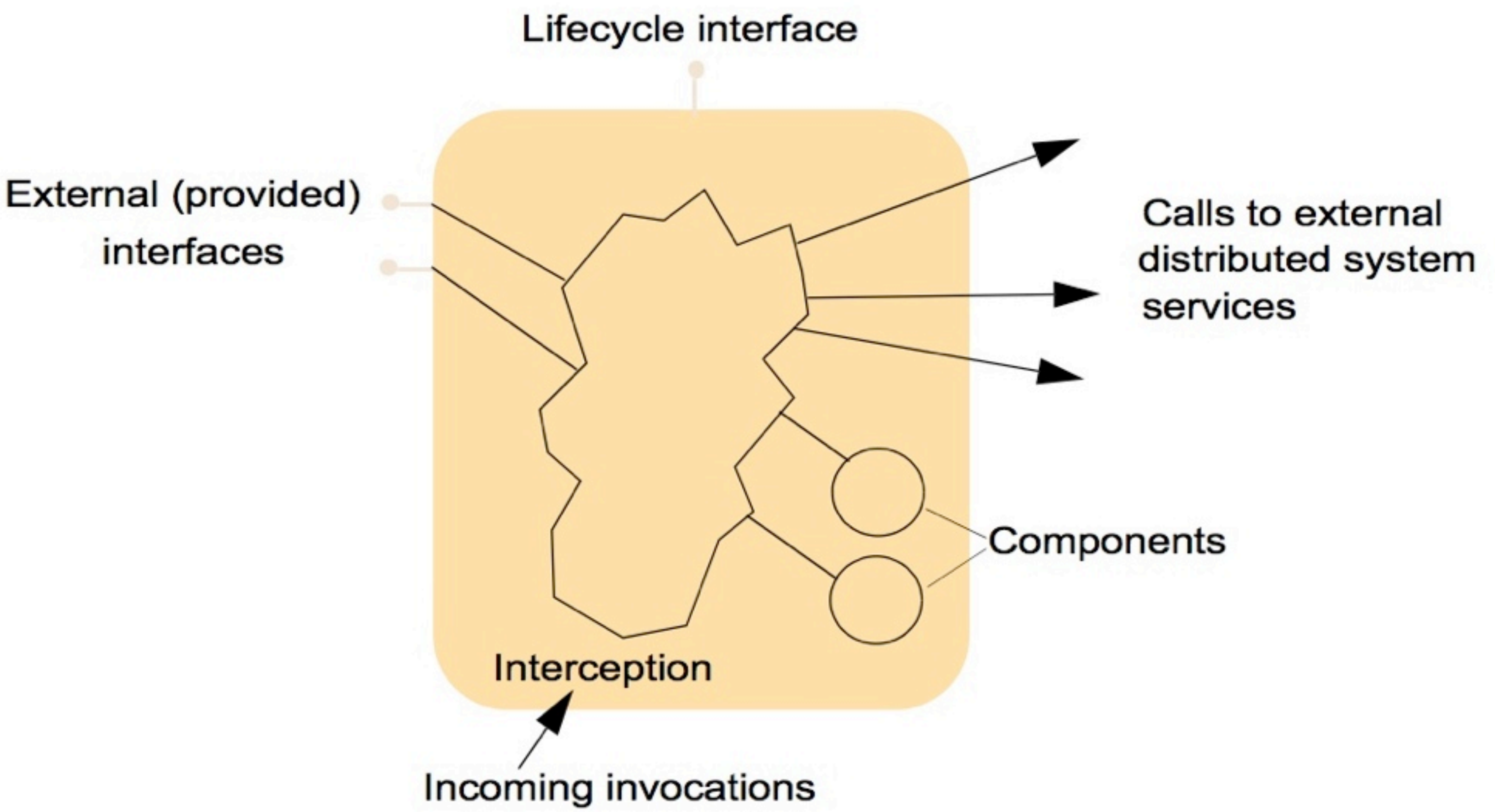
It consists of:

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

Tasks of a container

- Provides a managed server-side hosting environment for components
- Provides the necessary separation of concerns that means the components deal with the application concerns and the container deals with the distributed systems and middleware issues

The structure of a container



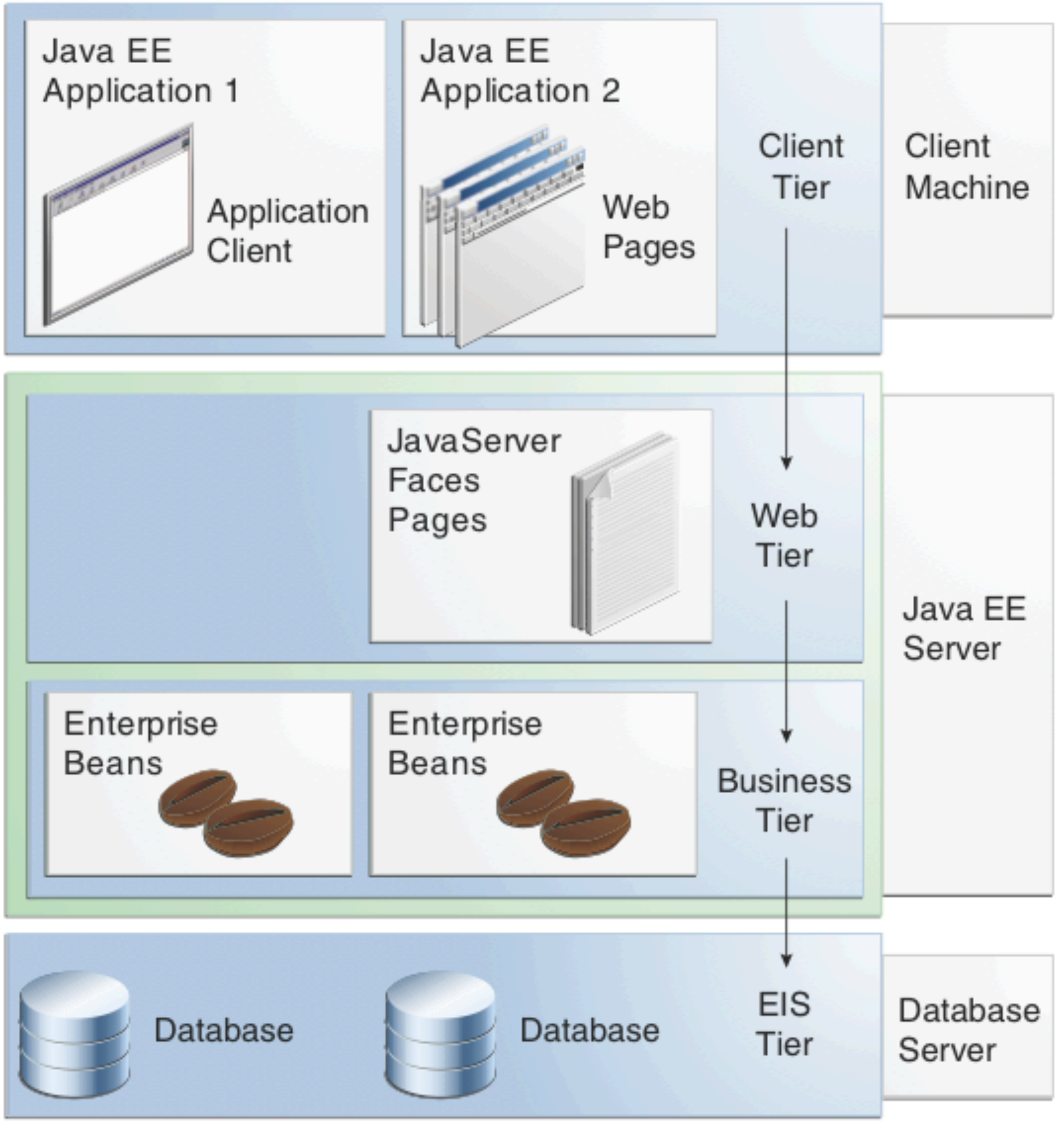
Application servers

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]

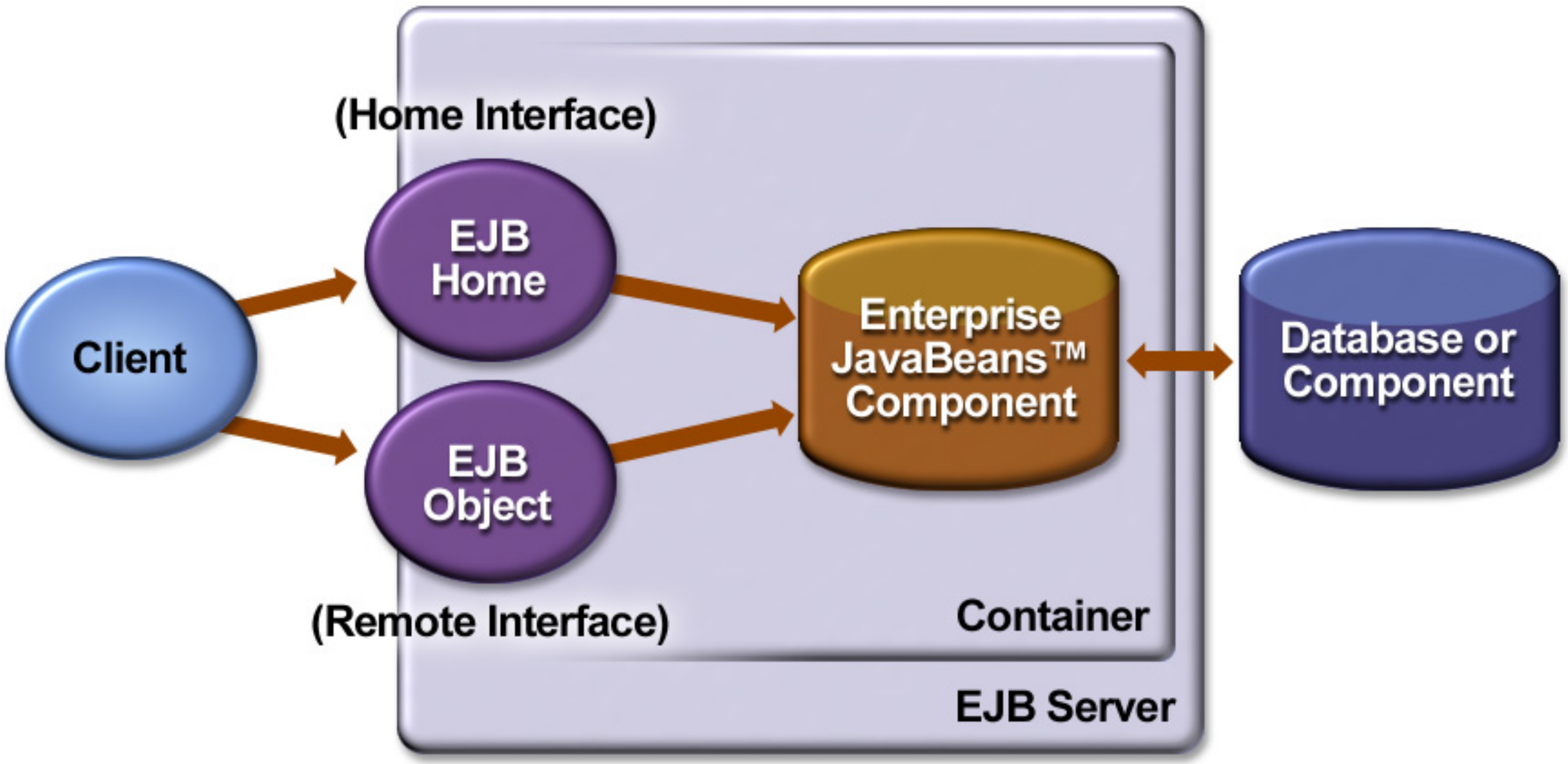
Distributed objects and components

Enterprise JavaBeans

Multi-tiered Java EE applications



EJB Architecture



Enterprise beans

The Enterprise JavaBeans architecture is a component architecture for the development and deployment of component-based distributed business applications.

Example: In an inventory control application, the enterprise beans might implement the business logic in methods called checkInventoryLevel and orderProduct.

Benefits of Enterprise Beans

- EJB container provides system-level services to enterprise beans, the bean developer can concentrate on solving business problems.
- Client developer can focus on the presentation of the client.
- Application assembler can build new applications from existing beans.

When shall I use enterprise beans?

- The application must be scalable. To accommodate a growing number of users, you may need to distribute an application’s components across multiple machines. Not only can the enterprise beans of an application run on different machines, but also their location will remain transparent to the clients.

- Transactions must ensure data integrity. Enterprise beans support transactions, the mechanisms that manage the concurrent access of shared objects.

- The application will have a variety of clients. With only a few lines of code, remote clients can easily locate enterprise beans. These clients can be thin, various, and numerous.

Types of EJBs

Session Bean: EJB used for implementing high-level business logic and processes

- Session beans handle complex tasks that require interaction with other components (entities, web services, messaging, etc.)

Timer Service

- EJB used for scheduling tasks

Message Driven Bean

- EJB used to integrate withe external services via asynchronous messages using JMS. Usually, delegate business logic to session beans

EJB containers

EJB container

- Runtime environment that provides services, such as transaction management, concurrency control, pooling, and security authorization.
- Historically, application servers have added other features such as clustering, load balancing, and failover.

Some JEE Application Servers

- GlassFish (Sun/Oracle, open source edition)
- WebSphere (IBM)
- WebLogic (Oracle)
- JBoss (Apache)
- WebObjects (Apple)

Distributed objects and components

Summary

So, what have we learned today?

- Be happy, this lecture is not part of the examination.

References

George Coulouris, Jean Dollimore, Tim Kindberg: *Distributed Systems: Concepts and Design*. 5th edition, Addison Wesley, 2011.

Michi Henning. 2008. The rise and fall of CORBA. Commun. ACM 51, 8 (August 2008), 52-57. DOI=10.1145/1378704.1378718 http://doi.acm.org/10.1145/1378704.1378718