DISTRIBUTED OBJECT AND COMPONENTS

Objectives

- Understand the development of distributed system
- Present middleware solutions for
  - Distributed object
  - Component based application
- Applications in a Java EE environment

Engineering problem

- How to build enterprise applications
  - Safe
  - Secure
  - Scalable
  - Available
  - Extensible
  - Cost effective

Fault tolerant systems

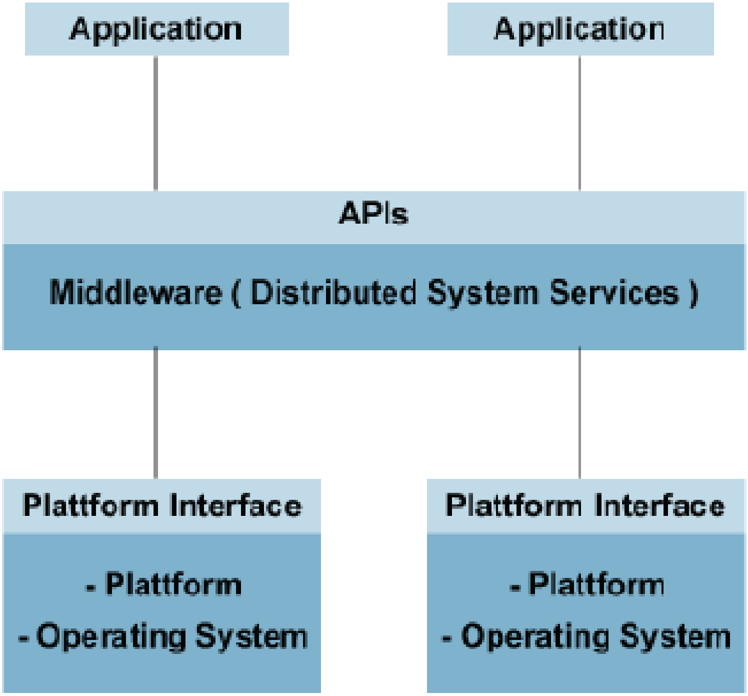
- The basic characteristics of fault tolerance require:
  - No single point of failure
  - Fault isolation to the failing component
  - Fault containment to prevent propagation of the failure
  - Availability of reversion modes
- Many kind of faults
  - Power
  - Network
  - Server
  - Disk
  - Virus or Attack
- Requires replication and redundancy

Failover / transparent failover

- What to do when a problem occurs
- How to make it transparent for the user
- Failover
  - Start a secondary system
  - Increase availability

Middleware

- Software between the operating system and the application

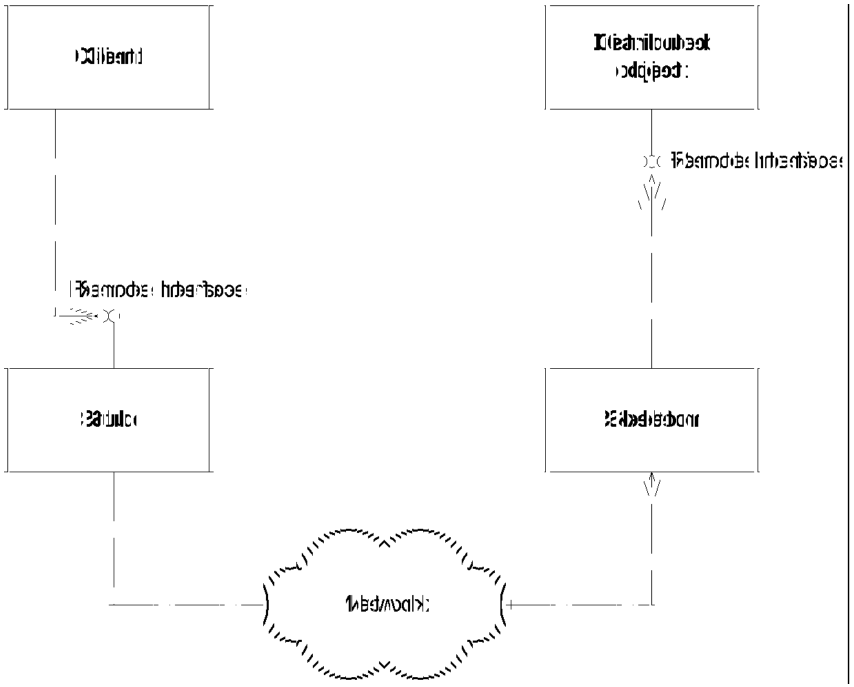


Le middleware

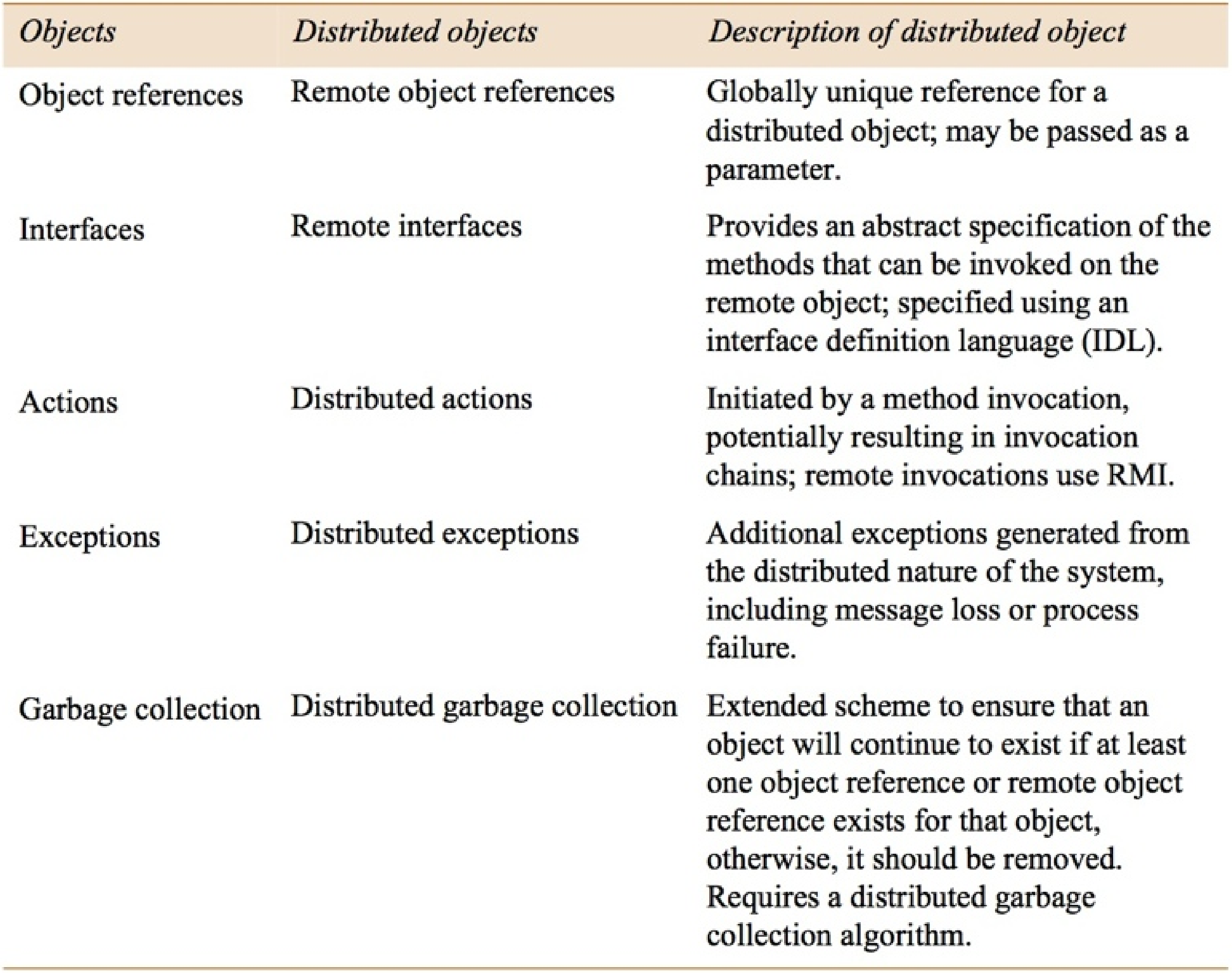
- Help to make distribution transparent
  - Transparency of heterogeneity
  - Transparency of location
  - Transparency of invocation
  - Transparency of transaction management
  - Transparency of security management
  - Transparency of service replication

Distributed Object

- A distributed object is made available to a distant client
  - Executing in another process,
  - From another place of the network



Distributed Object



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.

Developing a distributed object

- Take into account the execution context
  - Explicit middleware
  - The developer is in charge
- The environment manage the execution context
  - Implicit middleware
  - The container is in charge

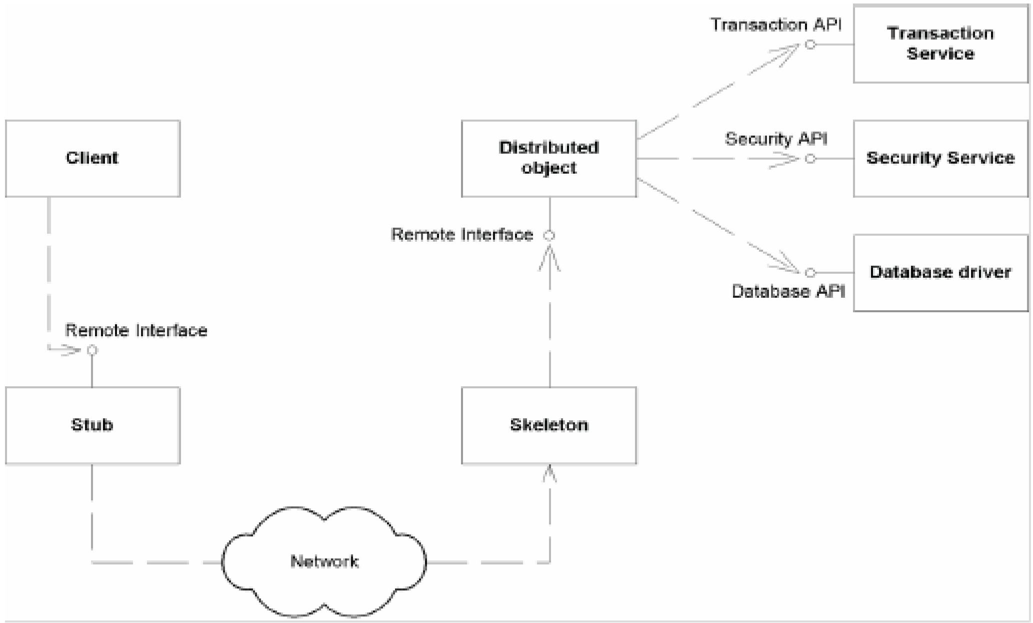
Software Code/Middleware code

- Software Code, related to the business
  - Management of a bank account
- Middleware code, not directly related to the business
  - Security management
  - Audit
  - Persistence management
- Goal : separate the application code from the middleware code

Middleware code

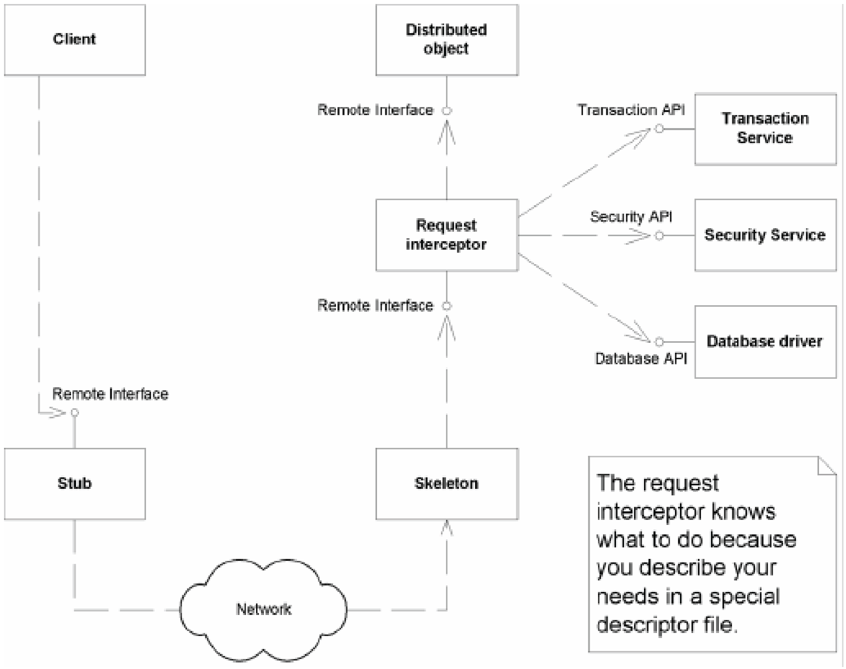
- Access as services
  - Persistence
  - Logging, audit
  - Transation
- Defined by API and policies
  - Start/commit transaction, store/load data...
- Question : how to manage both the middleware code and the application code

Explicit Middleware



```
transfer(Account account1, Account account2, long amount) {
// 1: Call middleware API to perform a security check
// 2: Call middleware API to start a transaction
// 3: Call middleware API to load rows from the database
// 4: Subtract the balance from one account, add to the other
// 5: Call middleware API to store rows In the database
// 6: Call middleware API to end the transaction
}
```

Implicit Middleware



The request interceptor knows what to do because you describe your needs in a special descriptor file.

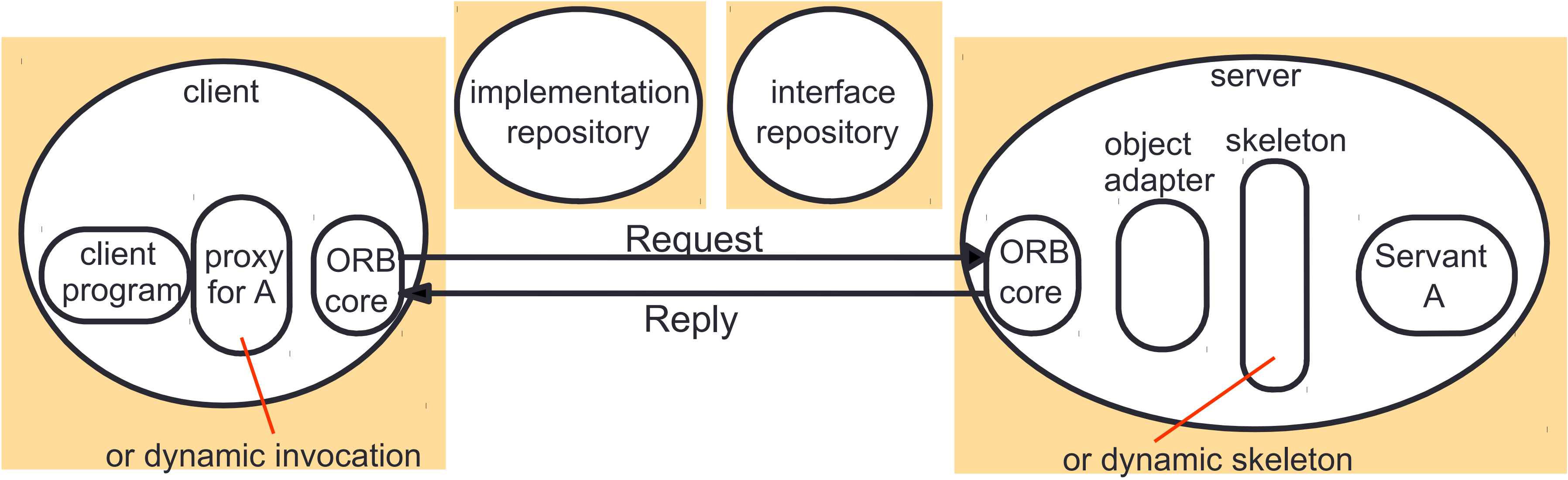
```
transfer(Account account1, Account account2, long amount) {
// 1: Subtract the balance from one account, add to the other
}
```

CORBA and object brokers

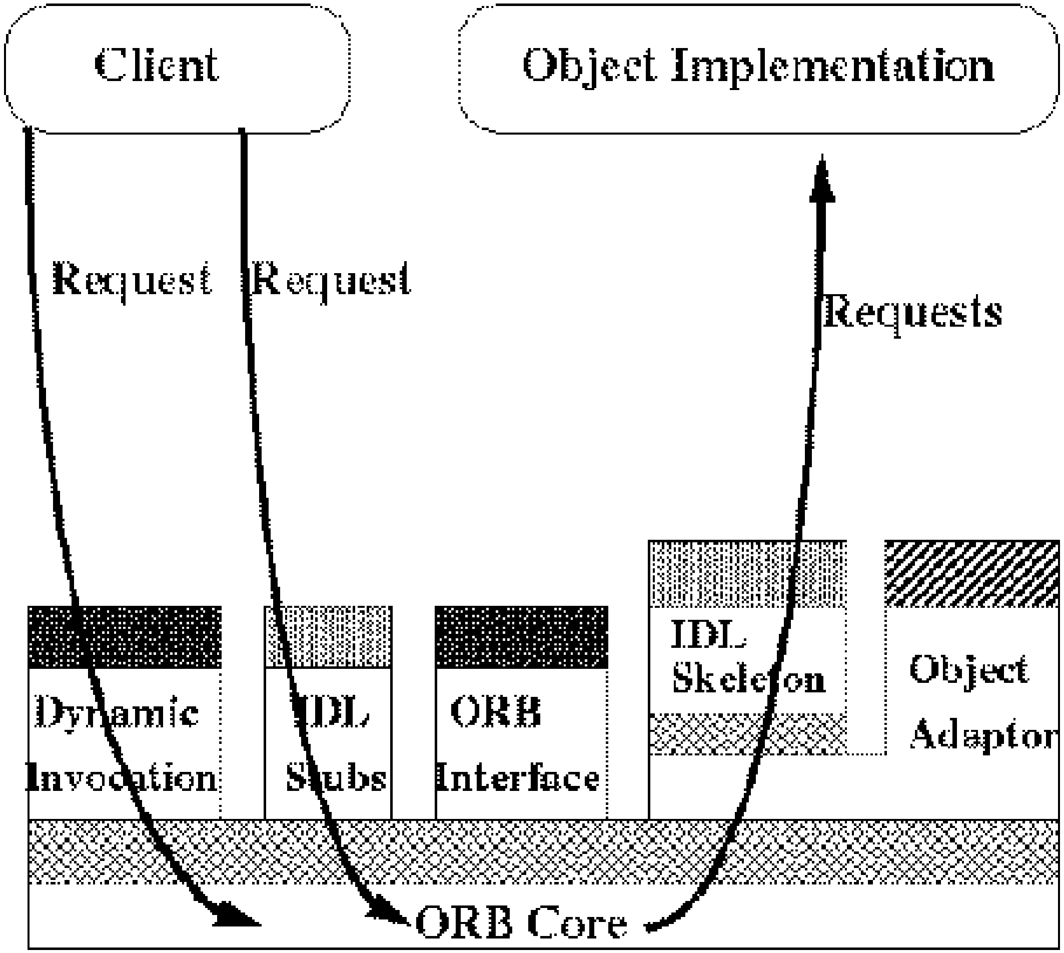
- Common Object Request Broker Architecture
  - www.corba.org
- Distributed Object Model
- Standardized by the OMG since 1990
- Objectives
  - Transparency of distribution
  - Interoperability

Object Request Broker

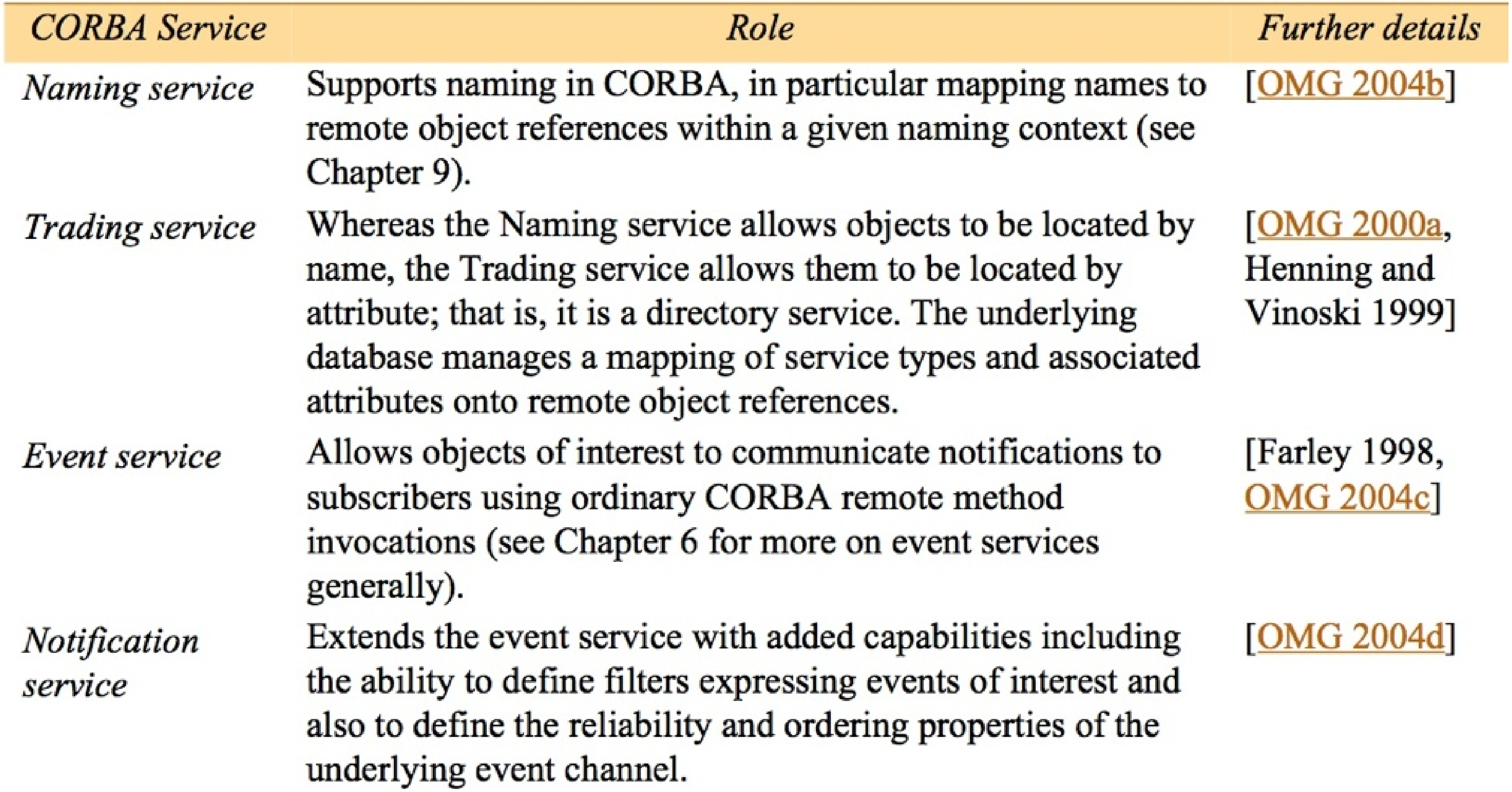
- Software bus to route request
- Services
  - Locate objects on the network
  - Call operation on the network
- Internet InterORB Protocol



The Broker Architecture

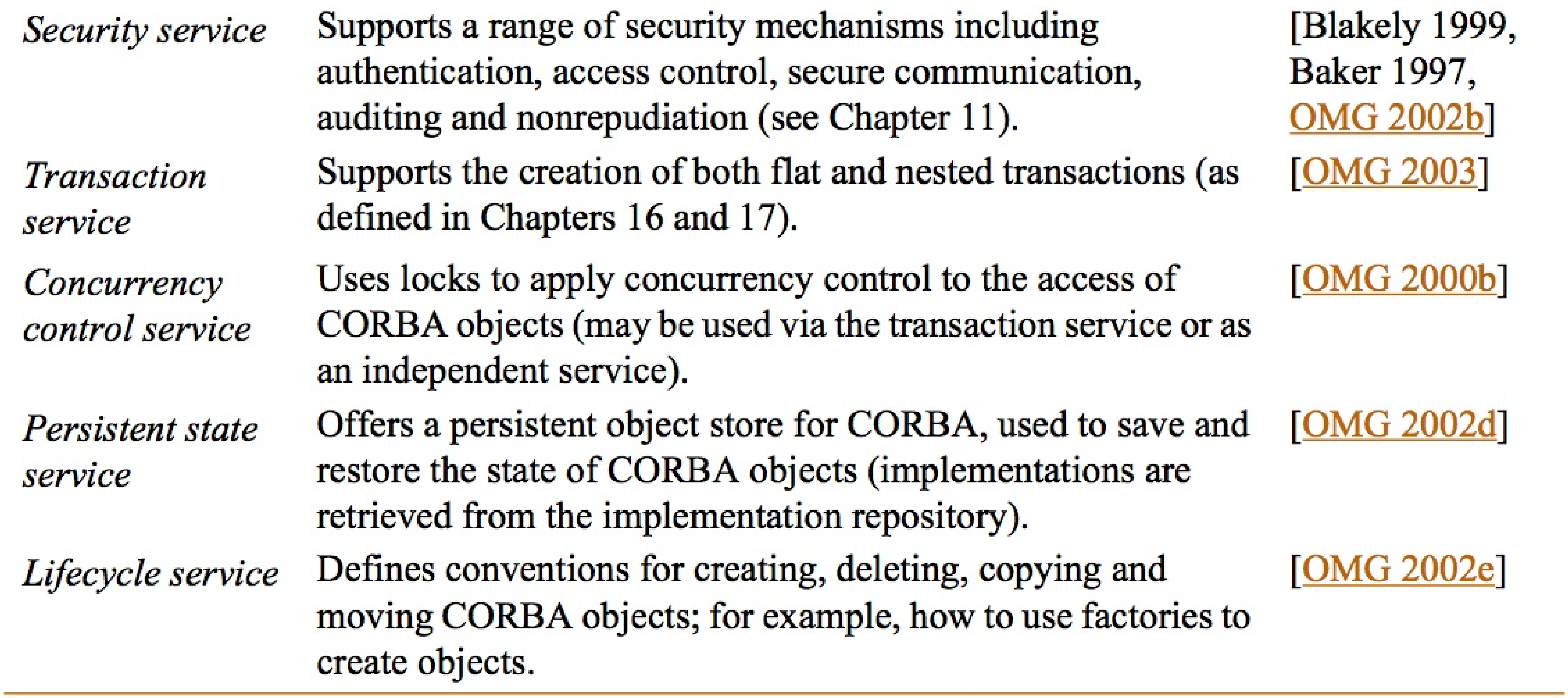


Corba services (1)



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]

Corba Services (2)



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]

CORBA and interoperability

- Agnostic language to define interface
  - IDL = Interface Définition Language
- IDL Translation from and to C++, Java, Python,C,…
- Allows to call a Java service from a C++ applications

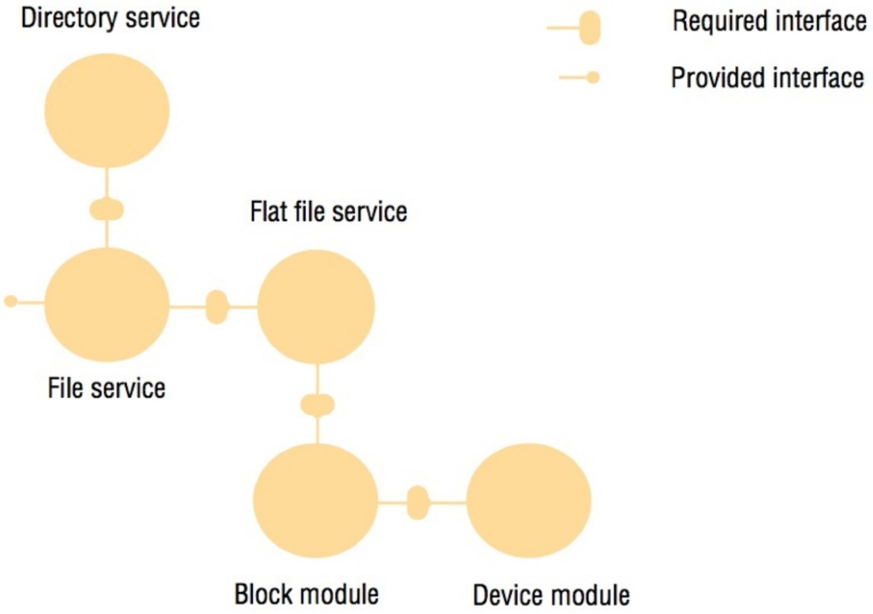
- No interoperability between ORB from different vendors

Problems with Distributed Objects

- Implicit dependencies
- Interaction with the middleware
- Lack of separation of distribution concern
- No support for deployment

Component

- A component complies to a contract to execute itself in a container
- Includes
  - A set of provided interfaces
  - A set of required interfaces

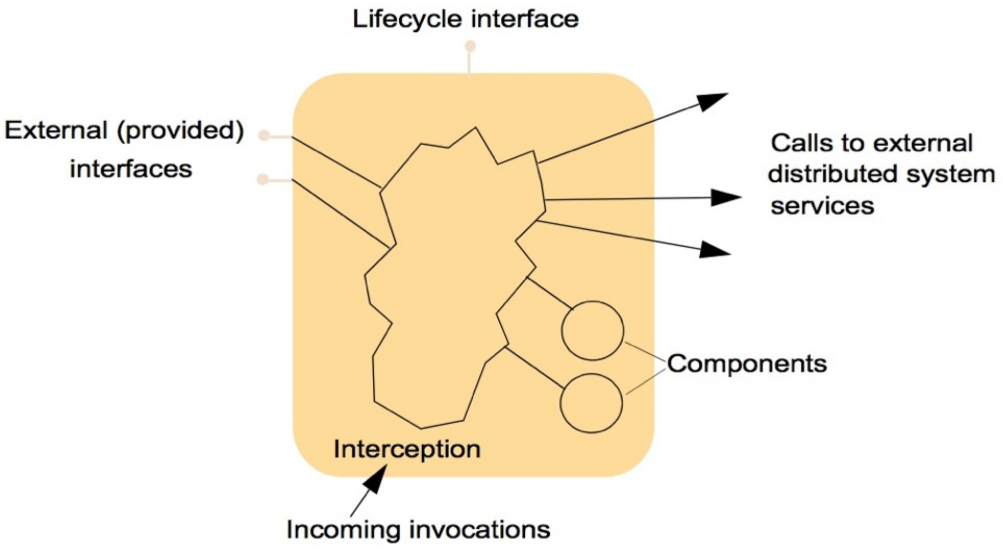


Definitions

- Council et Heinmann
  - A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard
- Szyperski
  - A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third-parties.

Container

- Provide an execution environment for component
- Manage the component life cycle
- Manage the relation with other services


Lifecycle interface
External (provided) interfaces
Calls to external distributed system services
Components
Interception
Incoming invocations

Component Charateristic (Sommerville)

- Standardised
- Independant
- Composable
- Deployable
- Documented

Components as service provider

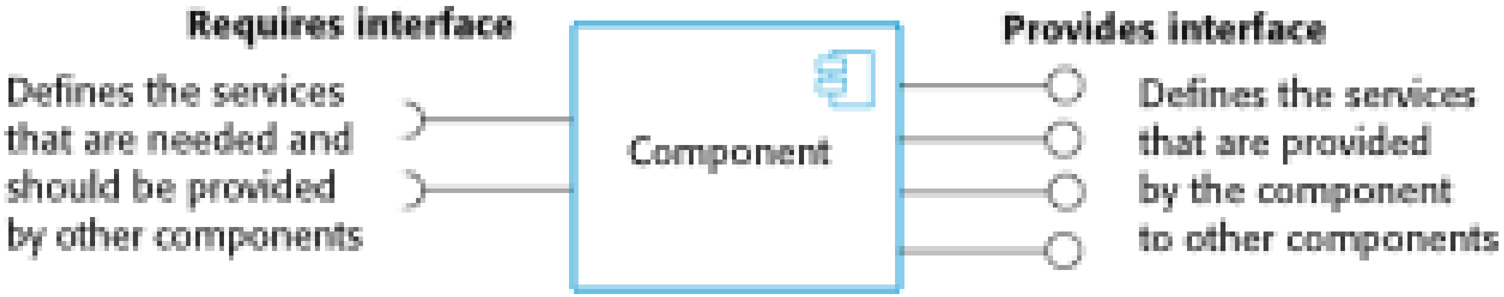
- Component are independant executable entities
- All interactions with a component take place through an interface
- The state of a component is never exposed

Interfaces

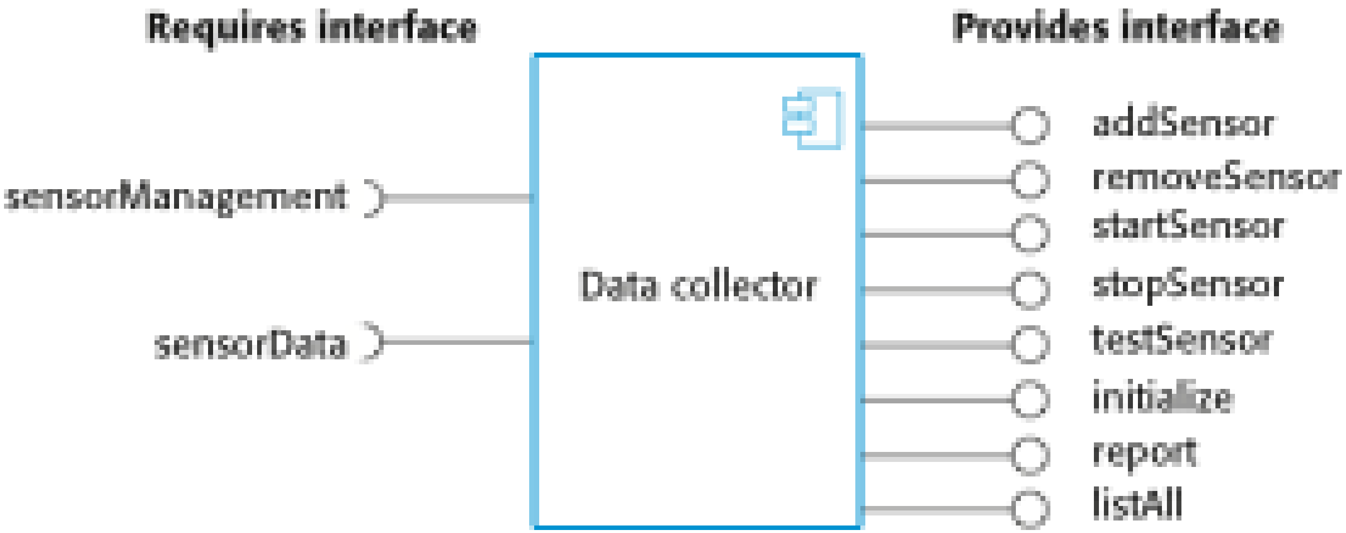
- Provides interfaces
  - Define the services provided by the component
  - API of the component
  - Used by the component client
- Required interfaces
  - Services needed by the component for its execution
  - No reference to how the service have to be provided

Interfaces

- UML notation


Requires interface
Defines the services that are needed and should be provided by other components
Component
Provides interface
Defines the services that are provided by the component to other components

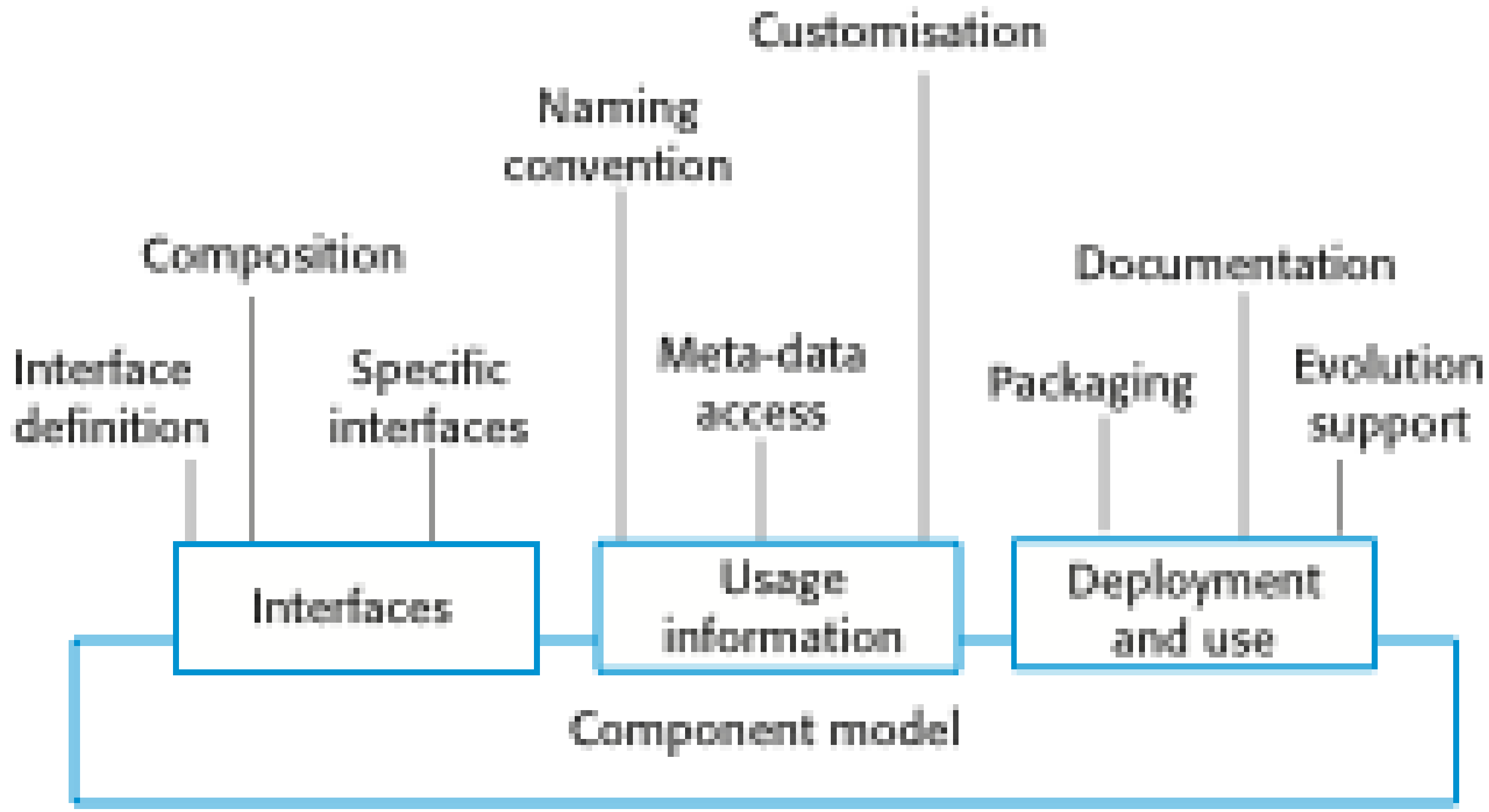
Example (from SE9)



Component models

- Definition of standards for the development, deploiement and documentation of components
- Examples
  - EJB
  - COM (.Net)
  - Corba Component Model
  - SCA

Basic elements



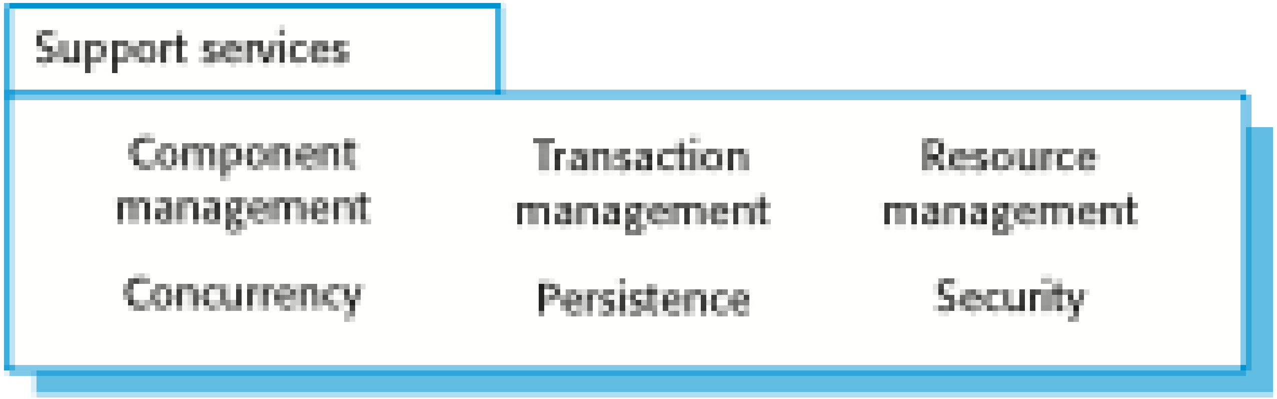
Component vs Objects

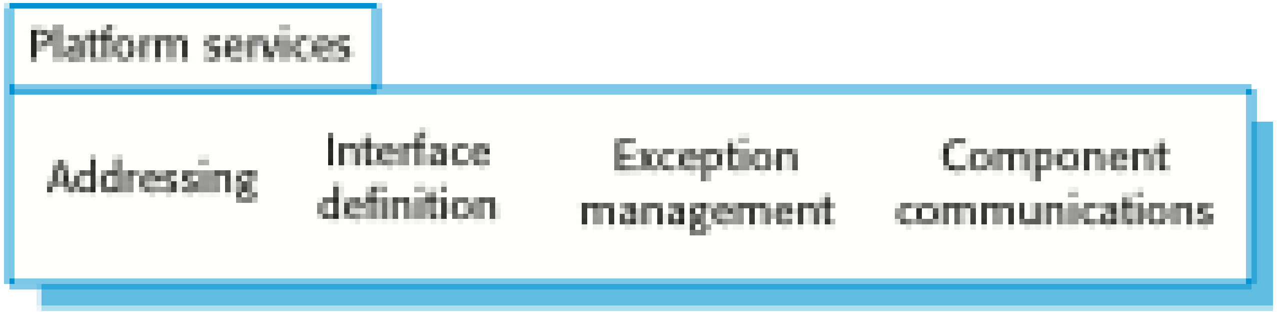
- Component can be deployed
- Component do not define type
- Component implementation is hidden
- Components are language neutral
- Components are standardized

Middleware support

- Component models are the basis for middleware that provides support for executing components.
- Component model implementations provide:
    - Platform services that allow components written according to the model to communicate;
    - Horizontal services that are application-independent services used by different components.
- To use services provided by a model, components are deployed in a container. This is a set of interfaces used to access the service implementations.

Middleware services in a component model

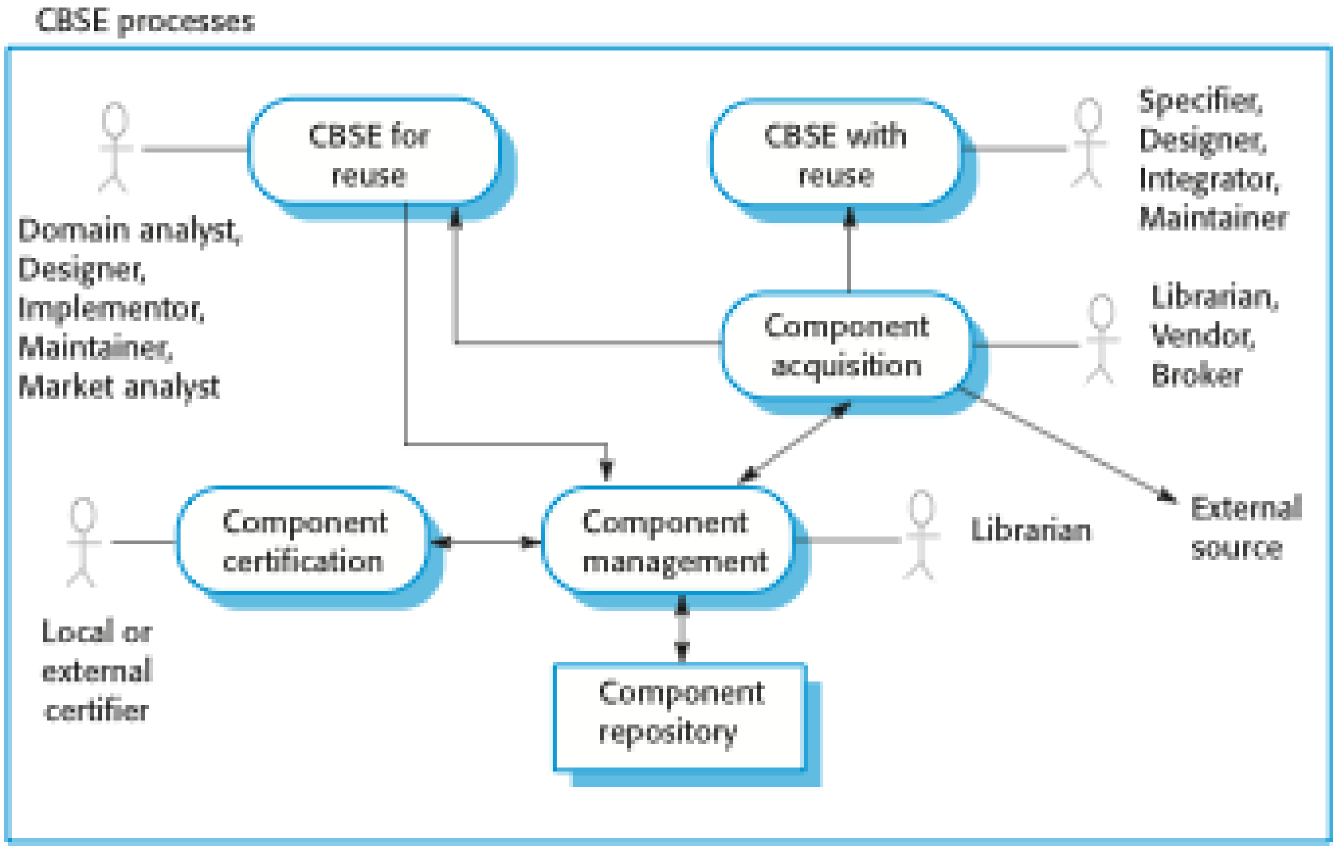




CBSE processes

- CBSE processes are software processes that support component-based software engineering.
    - They take into account the possibilities of reuse and the different process activities involved in developing and using reusable components.
- Development for reuse
    - This process is concerned with developing components or services that will be reused in other applications. It usually involves generalizing existing components.
- Development with reuse
    - This process is the process of developing new applications using existing components and services.

The Process


CBSE processes
CBSE for reuse
CBSE with reuse
Specifier, Designer, Integrator, Maintainer
Domain analyst, Designer, Implementor, Maintainer, Market analyst
Component acquisition
Librarian, Vendor, Broker
External source
Component certification
Component management
Librarian
Local or external certifier
Component repository

Supporting processes

- Component acquisition is the process of acquiring components for reuse or development into a reusable component.
  - It may involve accessing locally- developed components or services or finding these components from an external source.
- Component management is concerned with managing a company’s reusable components, ensuring that they are properly catalogued, stored and made available for reuse.
- Component certification is the process of checking a component and certifying that it meets its specification.

So why components

- Reuse
- NIH (Not Invented Here)
- High Cohesion/Low Coupling
- Explicit dependencies
- …

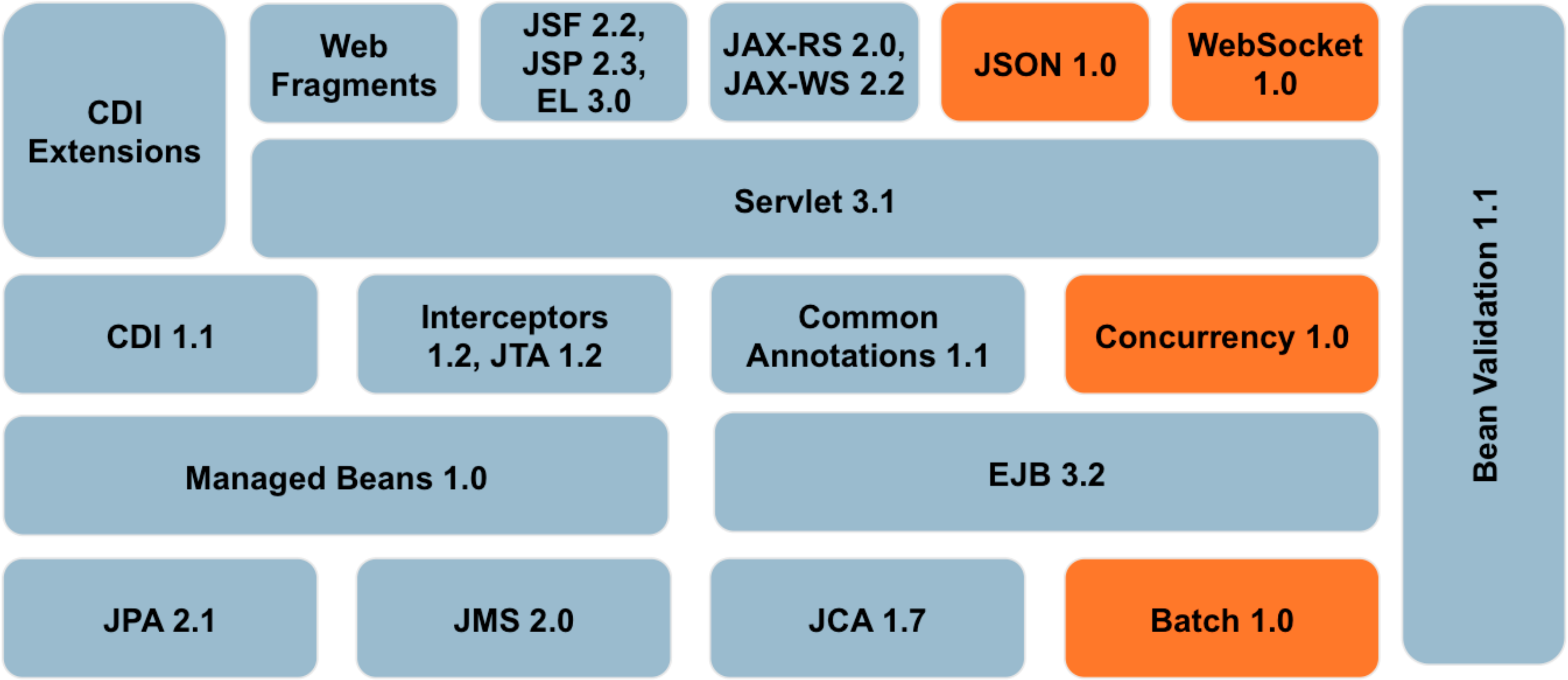
Key points

- CBSE is a reuse-based approach to defining and implementing loosely coupled components into systems.
- A component is a software unit whose functionality and dependencies are completely defined by its interfaces.
- A component model defines a set of standards that component providers and composers should follow.
- The key CBSE processes are CBSE for reuse and CBSE with reuse.

Java EE

- Application Server
- Define a stantard architecture including
  - A programming model (multi-tier, lightweight client)
  - A platform (ensemble de spécifications et de politiques requises)
  - A set of compatibility tests
  - A reference implementation
  - Patterns

Java EE 7 (2013)



New Stuff

- WebSOckets
- Batch applications
- JSON
- Concurrency
- New JMS API
- Transaction
- Default Resources
- More annotations
- Faces Flow (JSF)

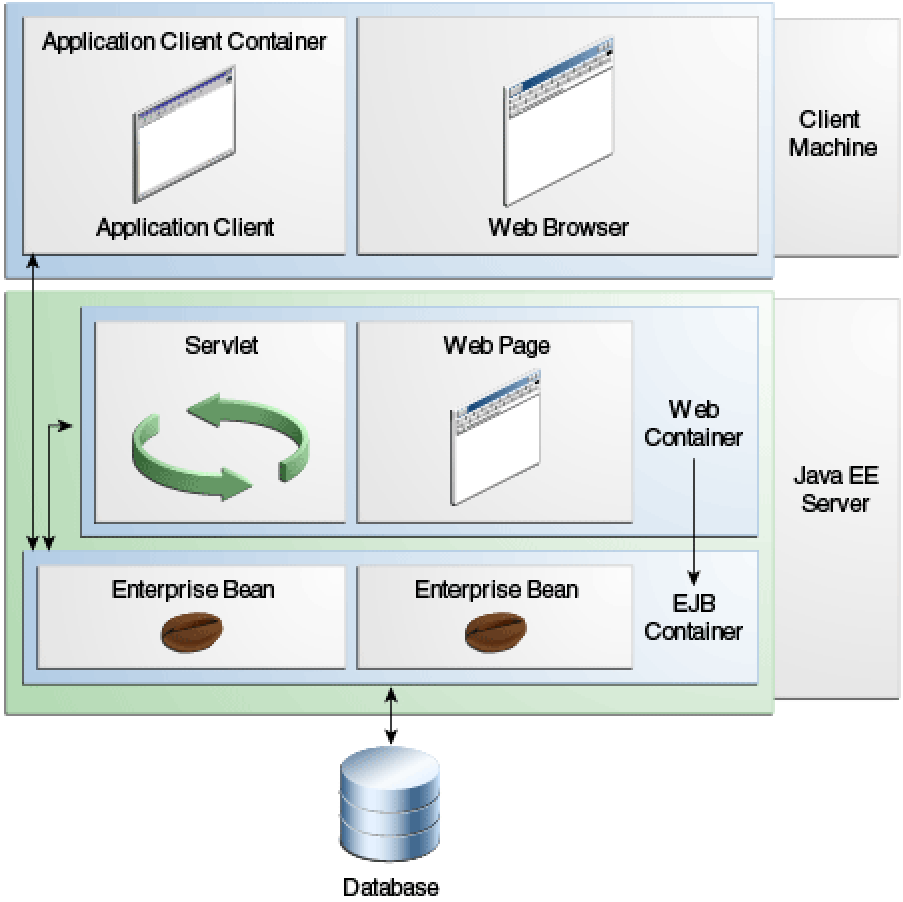
Application components

- 4 kinds of components
  - Client Component (Swing application)
  - Interface Component (Applet)
  - Web Component (Servlet, JSP, JSF)
  - EJB Component
- Component services
  - Management,
  - Deployment,
  - Execution

Containers

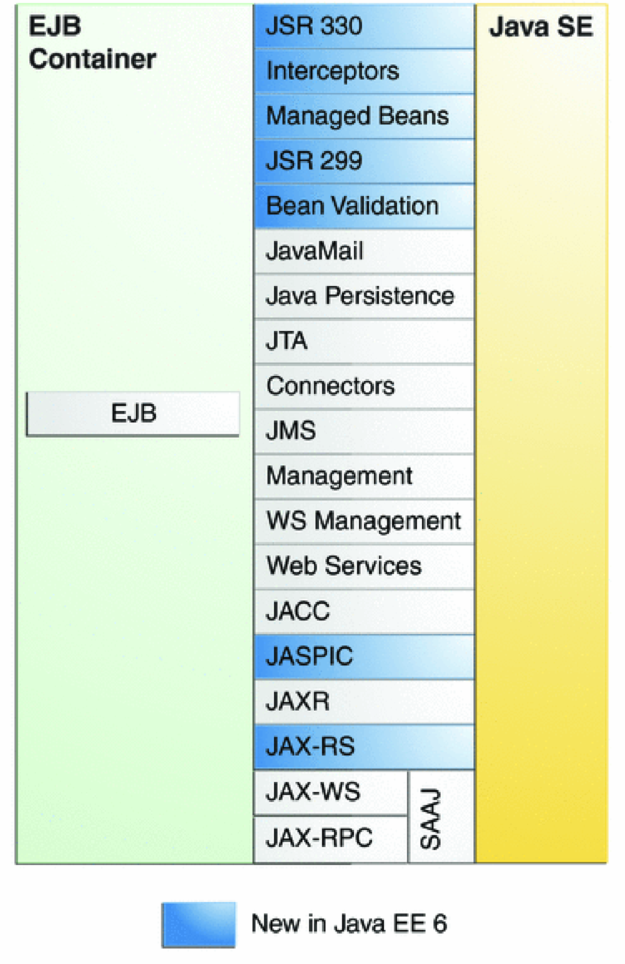
- Containers supports the executions of Java EE Components
- 4 Containers Types
  - ?

Containers types



From jee7 tutorial - https://docs.oracle.com/javaee/7/tutorial/overview004.htm /

Java EE Services



EJB Container

EJB

JSR 330

Interceptors

Managed Beans

JSR 299

Bean Validation

JavaMail

Java Persistence

JTA

Connectors

JMS

Management

WS Management

Web Services

JACC

JASPIC

JAXR

JAX-RS

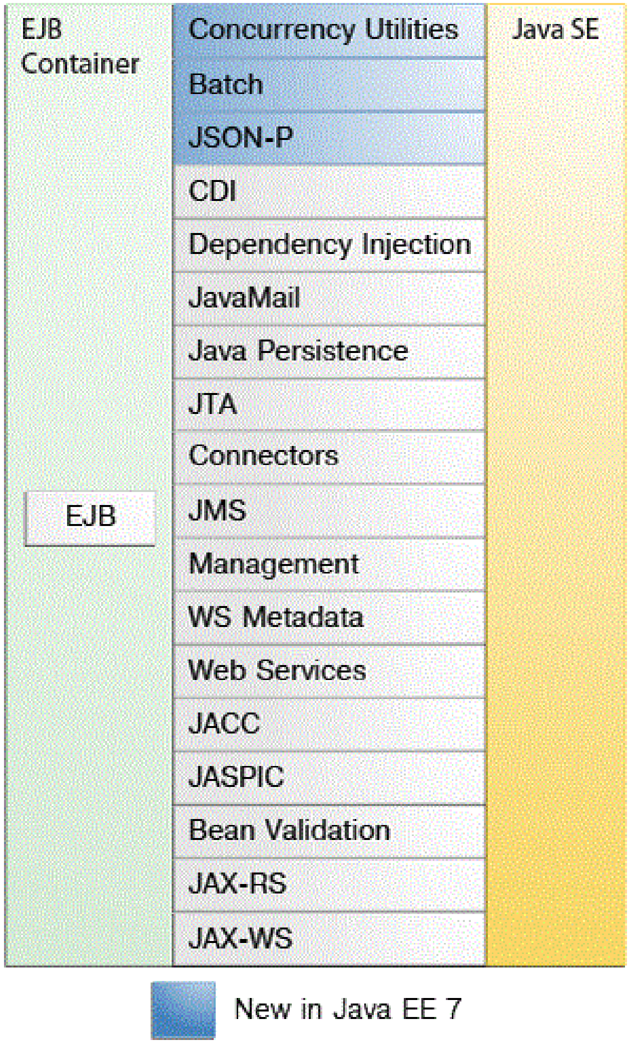
JAX-WS

JAX-RPC

SAAJ

Java SE

🟦 New in Java EE 6



EJB Container

EJB

Concurrency Utilities

Batch

JSON-P

CDI

Dependency Injection

JavaMail

Java Persistence

JTA

Connectors

JMS

Management

WS Metadata

Web Services

JACC

JASPIC

Bean Validation

JAX-RS

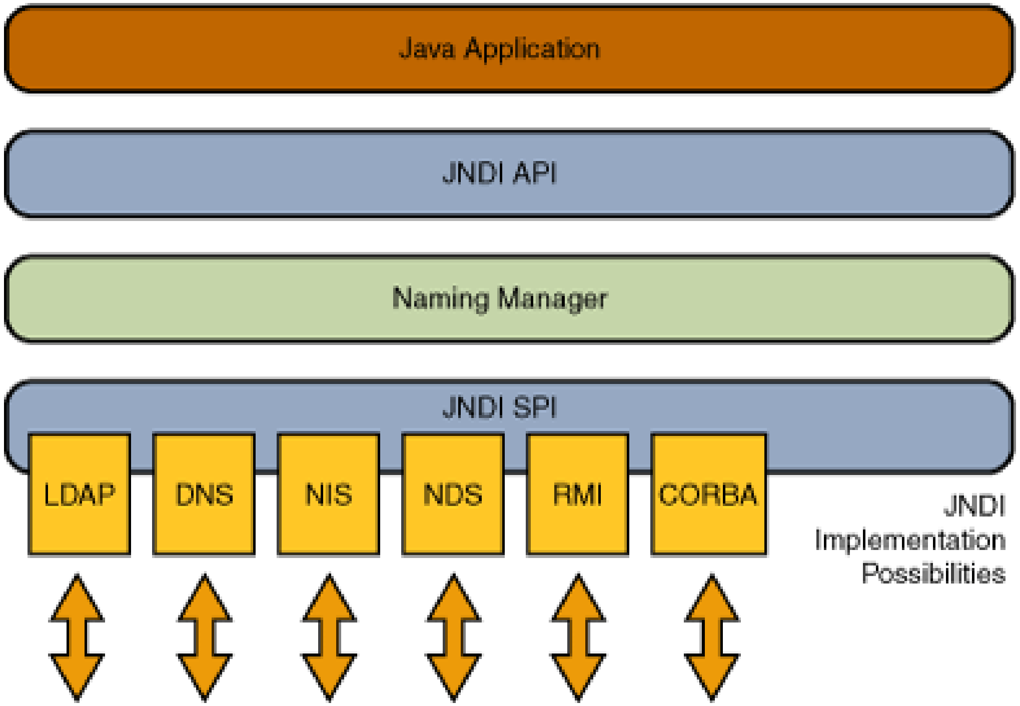
JAX-WS

Java SE

🟦 New in Java EE 7

JNDI (part of Java SE)

- Java naming and directory interface
- Naming Services



Java Application

JNDI API

Naming Manager

JNDI SPI

LDAP

DNS

NIS

NDS

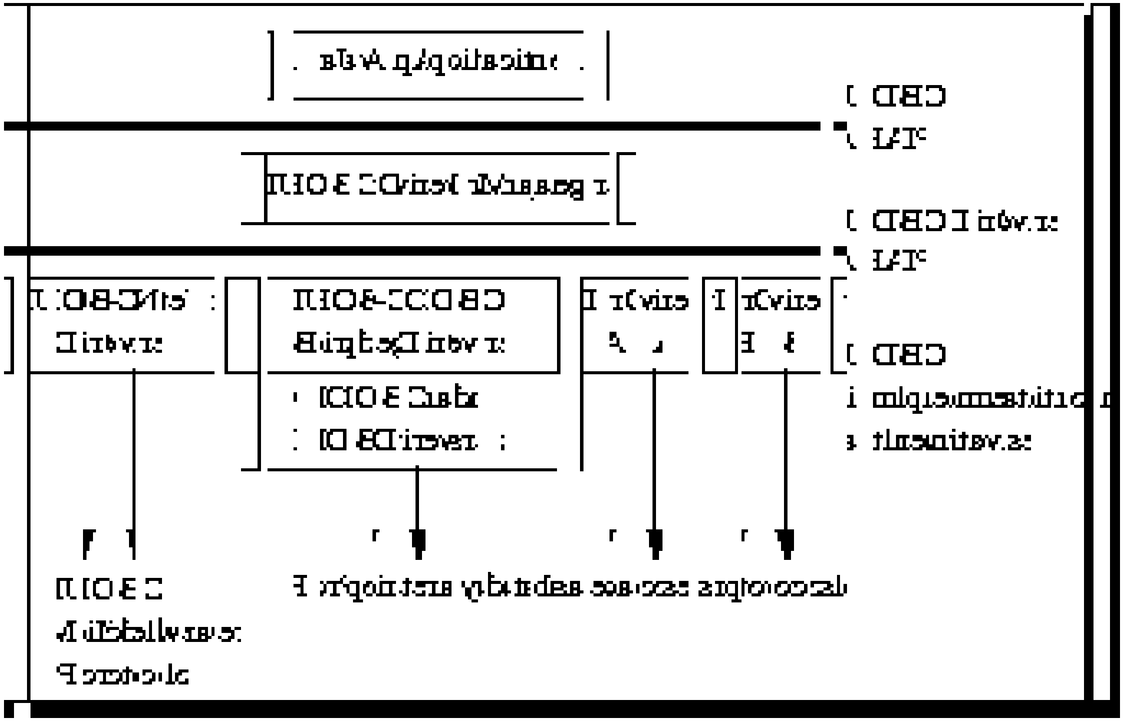
RMI

CORBA

JNDI
Implementation
Possibilities

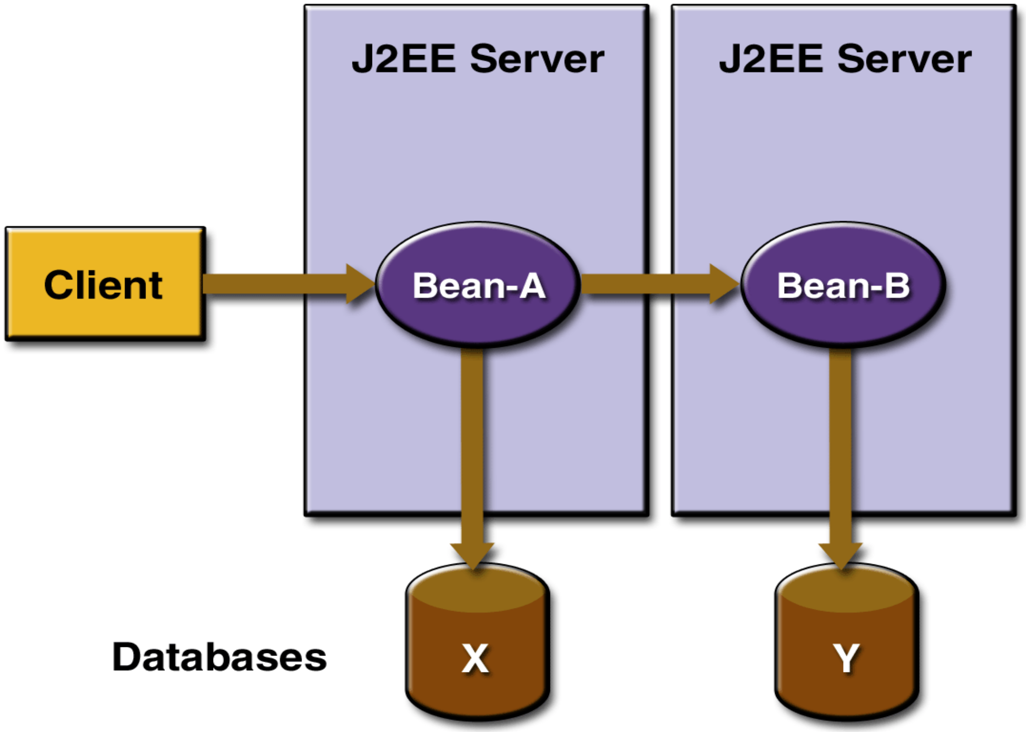
JDBC (Java SE)

- Java Database Connectivity
  - Relational Database Service

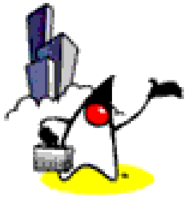


JTA

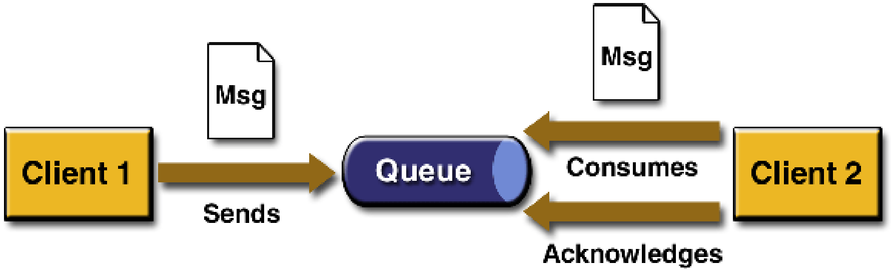
- Java Transaction Service
- Distributed Transaction Service

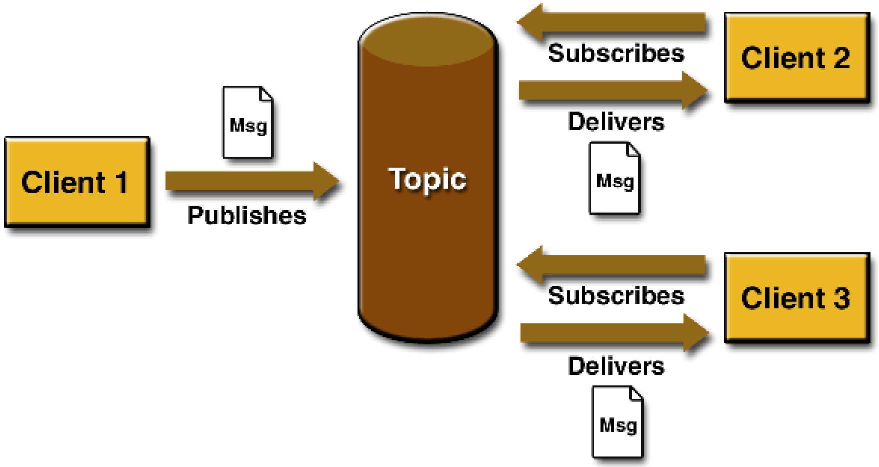


JMS



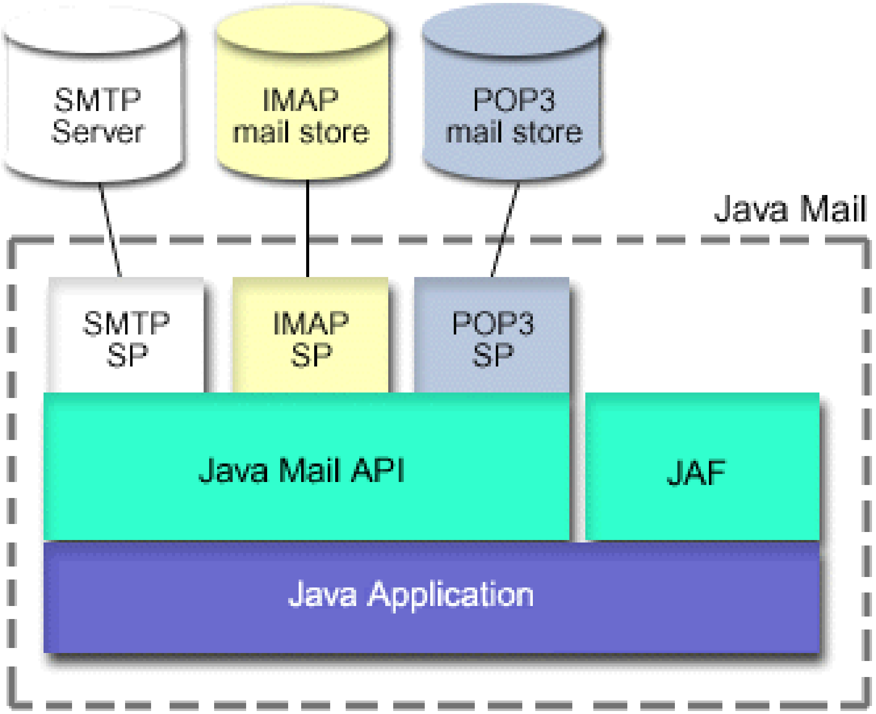
- Java Messaging Service
- Messaging Middleware





Java Mail

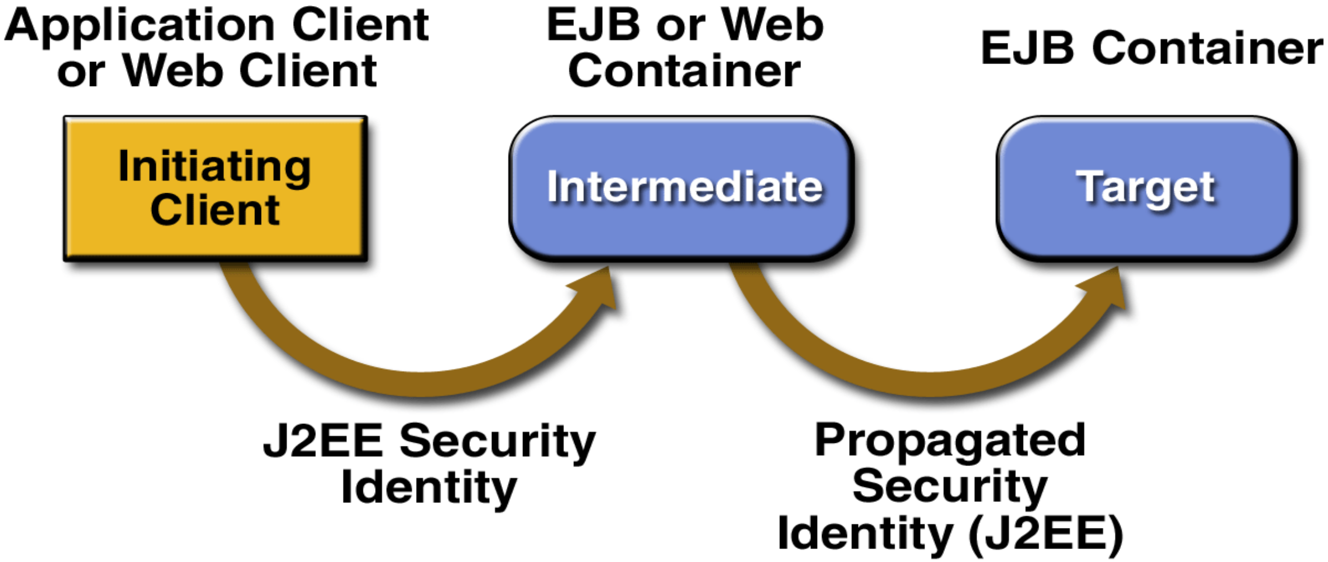
- Email Management Service



JASPIC and JACC

Java™ Security

- JavaTM Authentication Service Provider Interface for Containers
- Java Authorization Service Provider Contract for Containers
- Complement JAAS (from Java SE)



Application Client or Web Client

EJB or Web Container

EJB Container

Initiating Client

Intermediate

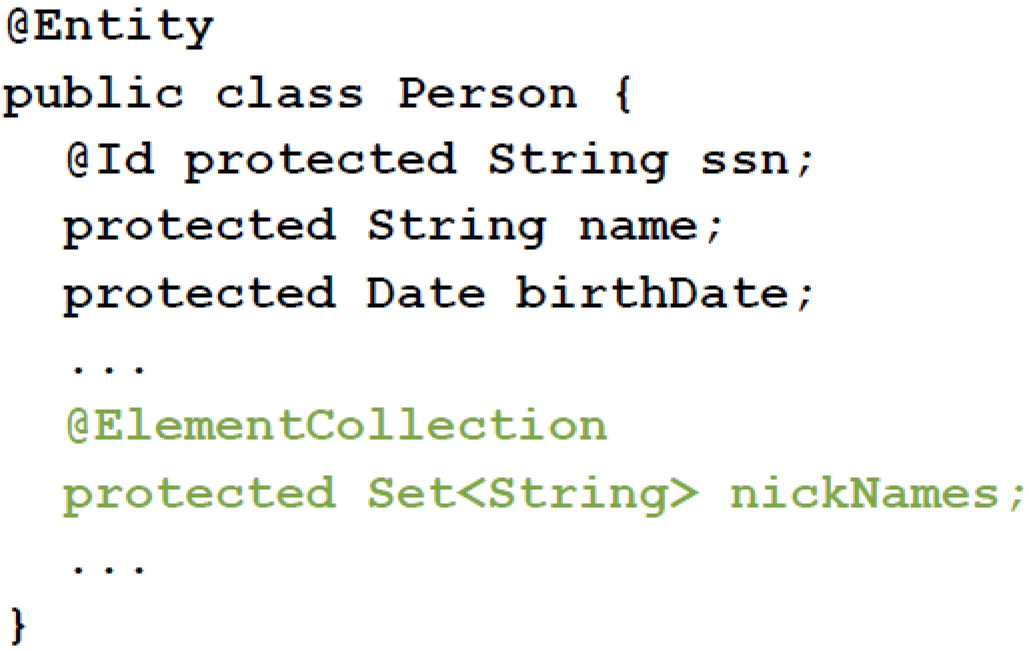
Target

J2EE Security Identity

Propagated Security Identity (J2EE)

JPA

- Persistence Service



```
@Entity
public class Person {
  @Id protected String ssn;
  protected String name;
  protected Date birthDate;
  ...
  @ElementCollection
  protected Set<String> nickNames;
  ...
}
```

The future of Java EE

- CF Java One 2016