Distributed Business Components

From objects to components

Overview

1. Part One: General Component Technology
    - What's wrong with objects?
    - Distributed Components
    - Business Concept mapping

2. Part Two: Enterprise Java Beans Example
    - Object Model
    - Basic Mechanisms
    - Separation of Concerns: Persistence, Transactions, Security
    - Separation of Context: Environment
    - Evolution and Lessons Learned

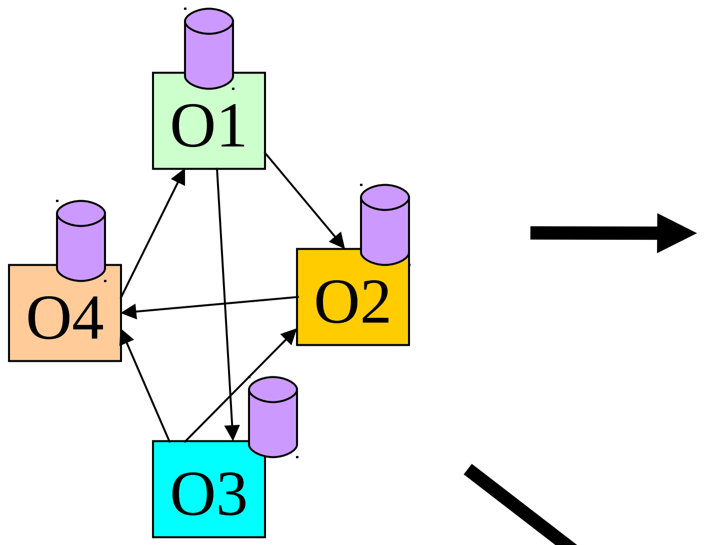
Part One: General Component Technology

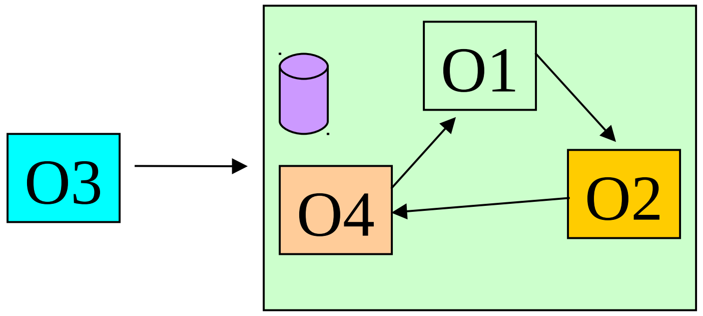
The big problems of Remote Objects

- Interfaces: too granular and therefor slow
- Modeling: The idea of "Business Objects" with internal workflow behind methods never worked
- No security support
- No persistence support
- No transaction support

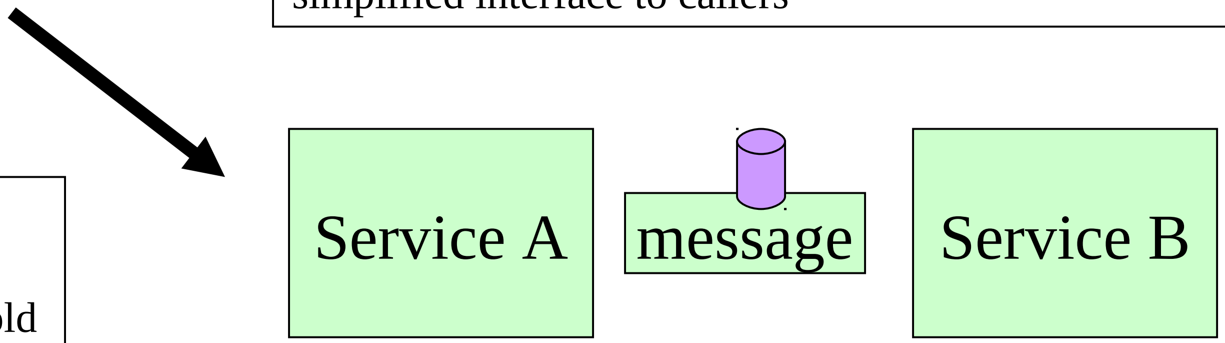
CORBA created services to address those issues. Java EE went the component and framework way. To work, the object idea needs state and state management!

What‘s wrong with Object Interfaces?





A component framework encapsulates objects and offers a simplified interface to callers



Object interfaces are tightly intertwined networks of references (links). Nodes hold state and link information for calling nodes. Caller and callee share state and promises. Requirement changes cause ripple effects and round trip times are enormous.

A messaging system can be stateless or include all state in the message itself (context complete communication, Neward). Webservices and REST approaches follow this architecture which has less mutual responsibilities. The current buzzword is Service Oriented Architecture (SOA). The current concept of decoupling is called Enterprise Service Bus.

What makes component based processing?

- Components: self contained software „packages“ with runtime interface, automatic deployment (install), built to fit into component framework
- A component framework where components plug in.
- Support for composition of and collaboration between components
- A set of roles for development, composition and installation of components (who does what in the component model?)

Enterprise components:

- Integration into existing infrastructure (transactions, security, legacy systems) and requirements (scalability, customizable, maintenance)
- Network addressable interfaces
- Medium to large granularity (e.g. representing 10-20 tables!)
- Representing a business concept (isomorphically).

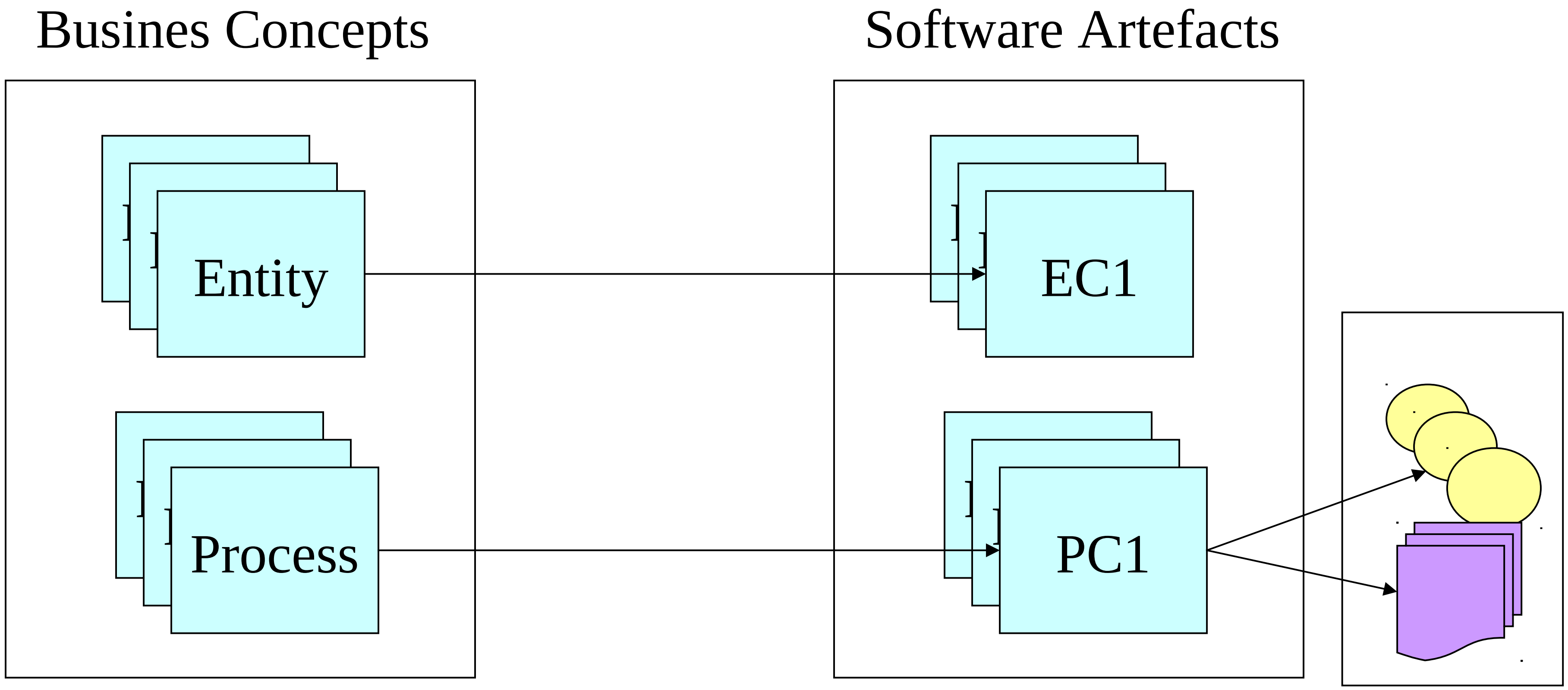
from Herzum, Sims, Conceptual framework chapter. We will compare EJBs later with these definitions. Especially the last one concerning the relation business concept – component and the granularity statement

from objects to components

dist. components	framework for pluggable business components. A market for interoperable components. Modeling, development and deployment are covered
dist. systems	multi-tier systems, Point-to-point connectivity with very complicated mechanisms. Expensive and hard to develop.
dist. objects	calls between applications. Management and performance problems with large numbers of small remote objects.
object-oriented	isolated, monolithic applications. No distribution

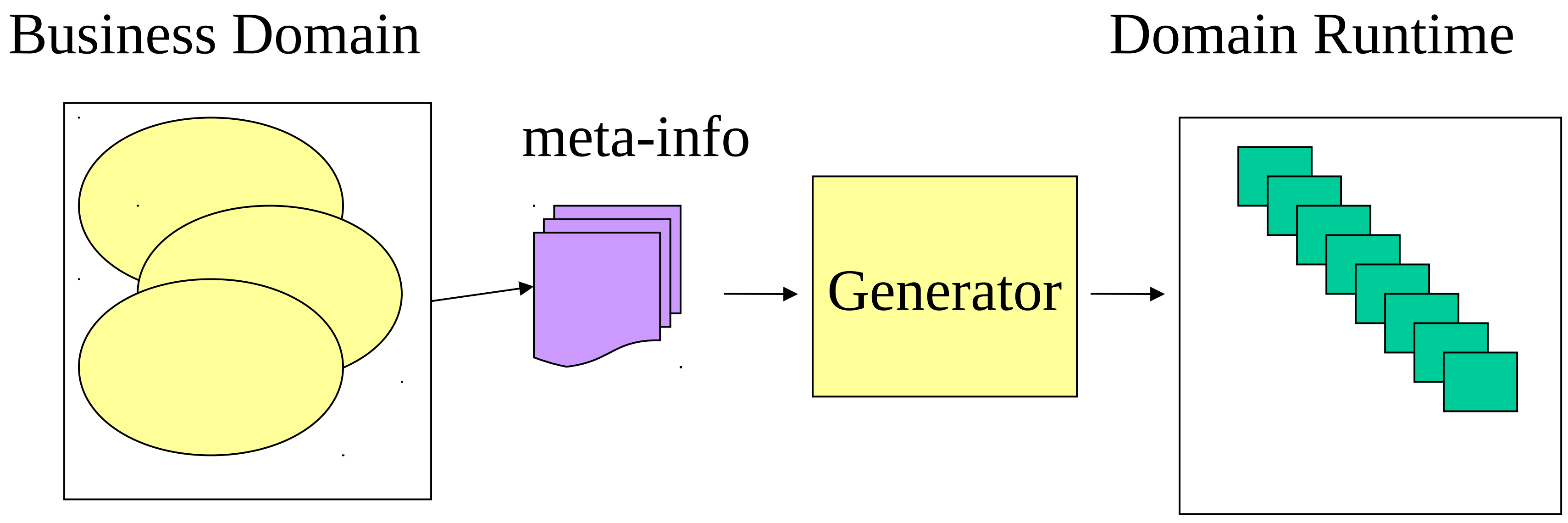
Note that components go beyond distributed systems to achieve re-use and a lower cost of development. And remember that these promises were already made for OO-based development!

Business Components



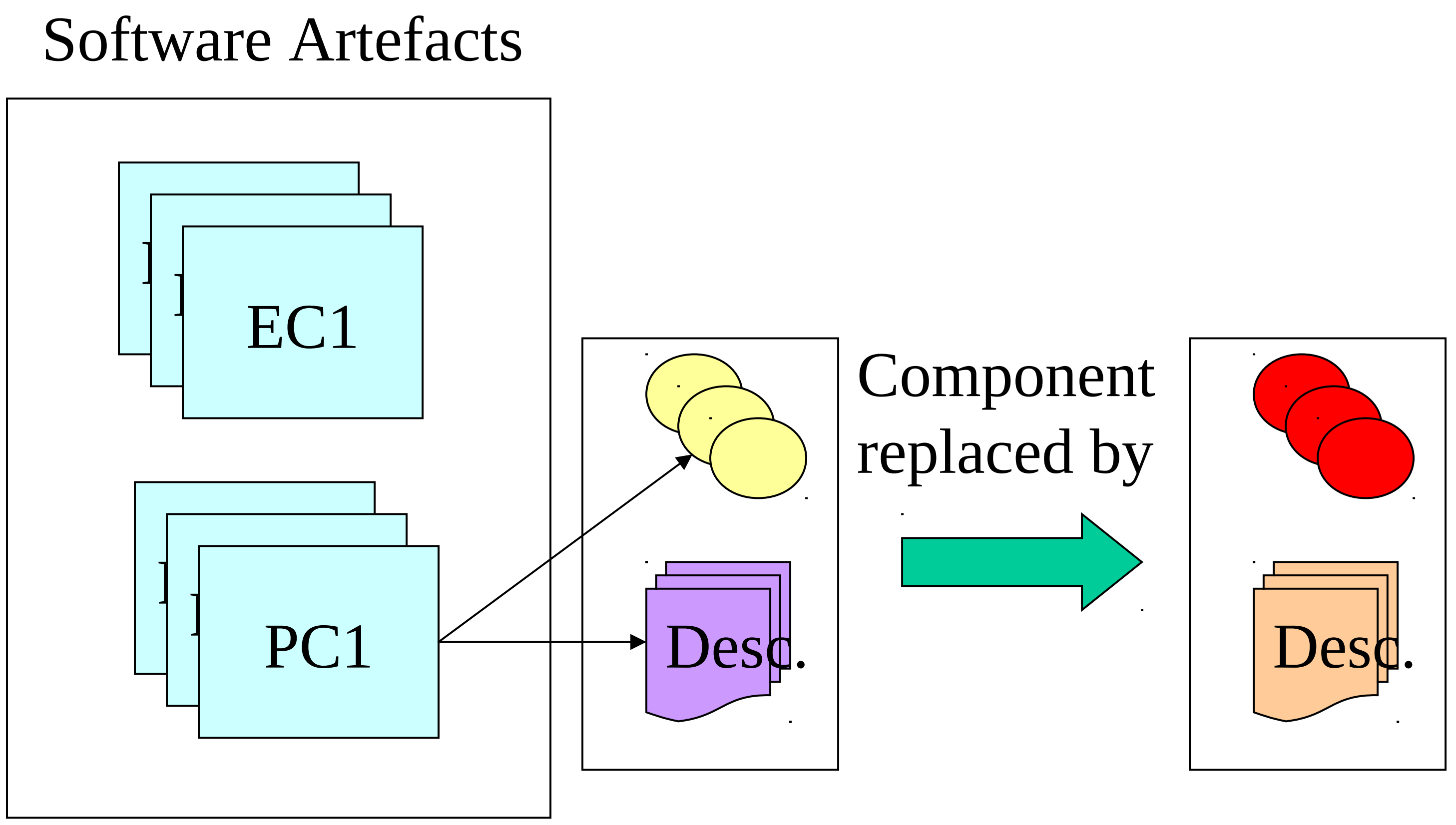
Business Components are supposed to directly represent concepts from the business,. The UML „package“ construct resembles the concept of business components.

Alternatives to the concept of isomorphic mapping



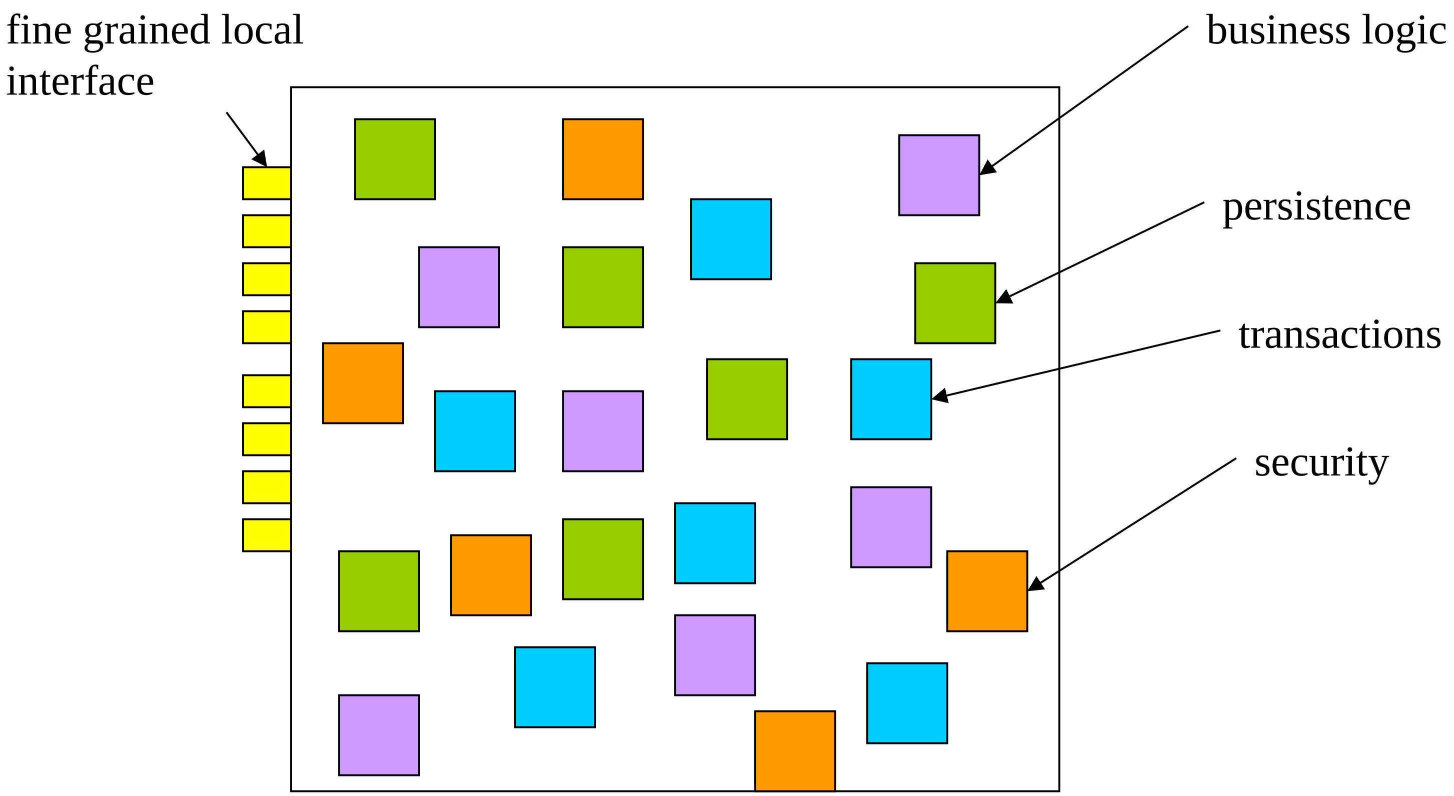
Do software artefacts really map isomorphically to business concepts? Istn‘t this a bit like trying to find the concept of „table“ by looking in our brains? Domain Analyses, generative computing and aspect oriented development are alternatives. In generative computing several views have been developed to capture the transformation process: computation independent model, platform independent model, platform dependent model.

Components: Code and Descriptors



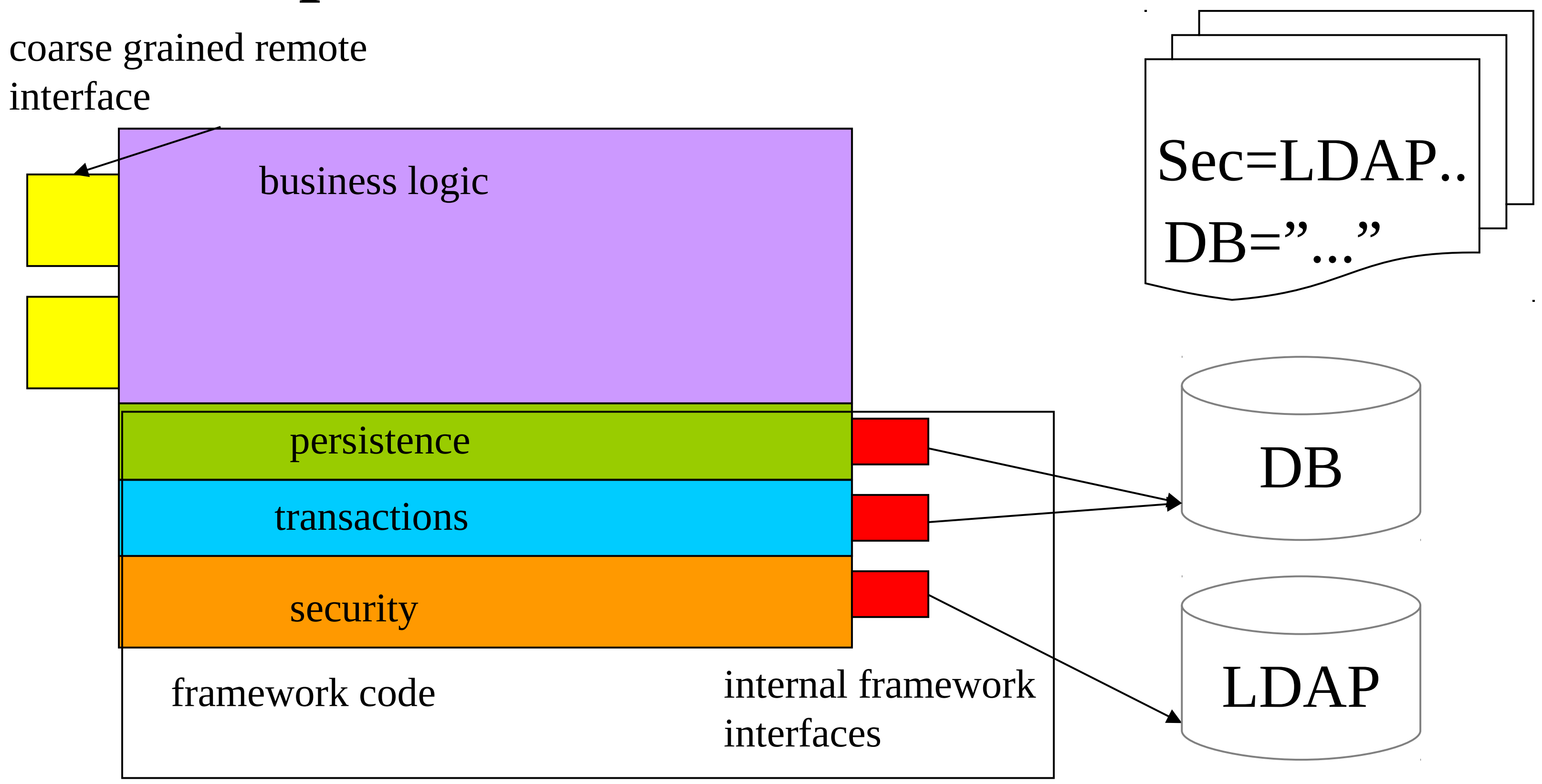
The components are clusters of software, configuration etc. which form a deployment and maintenance unit within the component framework. The software artefact „application“ has been replaced by a collection of collaborating components which can be adjusted without source code changes.

why objects are not components



a regular object mixes business logic with specific mechanisms (e.g. persistence) and hides the internal interfaces (implementation) behind the external interface. Assumptions about environment (e.g. which DB to store state in) are hidden in the code. Customizing objects means code changes.

Separation of Concerns and Context



the internal interface of a component is described in meta-information. Deployers can use this information to connect the component to the proper framework services. Concerns (e.g. persistence and transactions) are separated from business logic. Their implementation is typically done by the framework. Context information (which DB to use) is contained in meta-information and not in code. The component can be customized AFTER development.

Part Two: Component Technology Example EJB

Enterprise Java Beans

- allow construction of distributed applications by combining components from different vendors.
- developers need not understand low level distributed mechanisms (transactions etc.)
- components will run in EJB containers from different vendors unchanged
- Enterprise lifecycle support (development, deployment, runtime)
- Enterprise data support

Enterprise data are data that are a) important for the business and b) shared and used by many applications and users.

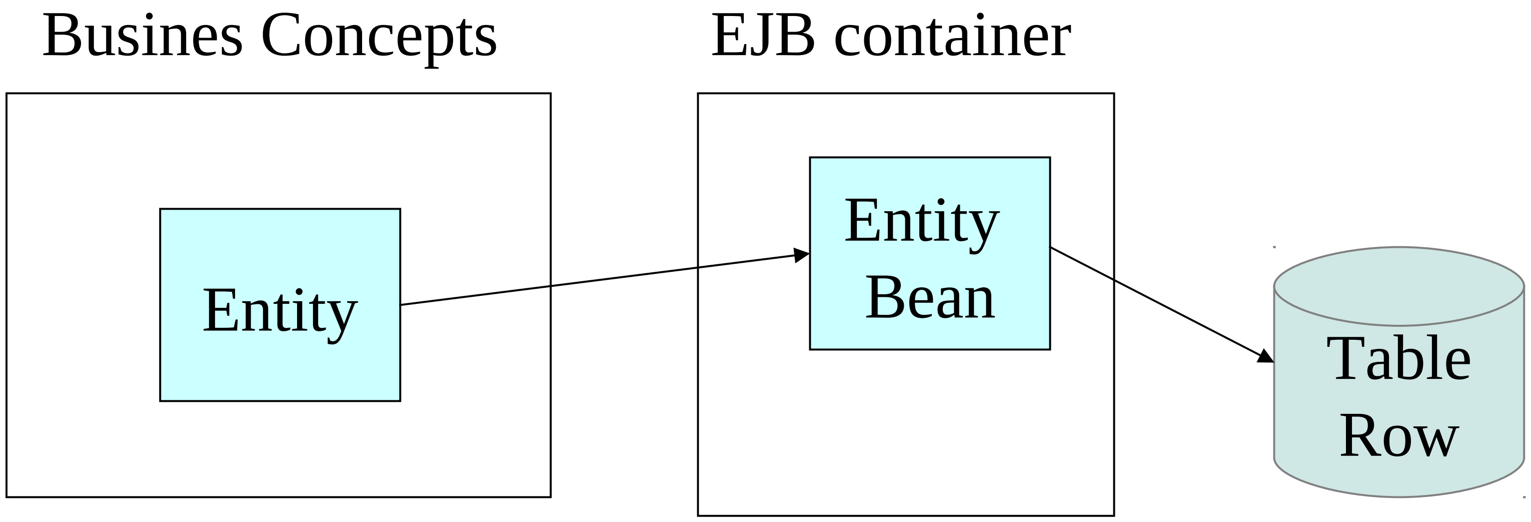
Enterprise Java Beans: Transactional Beans!

„Well, EJB is really all about transactional processing. I mean, at the end of the day, you go through the EJB spec and easily 60% of the entire spec is talking about transactional this, transactional that, transactional the other thing. Some of the remoting aspects, they're very quickly taken care of. Some of the life cycle aspects, very quickly taken care of. This is not hard stuff for them to do. „

„Enterprise Java Beans should never have been called Enterprise Java Beans, there's 101 EJBs damnation lists out there, and they said it best. It should not have been called Enterprise Java Beans, it should have been called Transactional Java Beans, because that's really what the spec focuses on. „

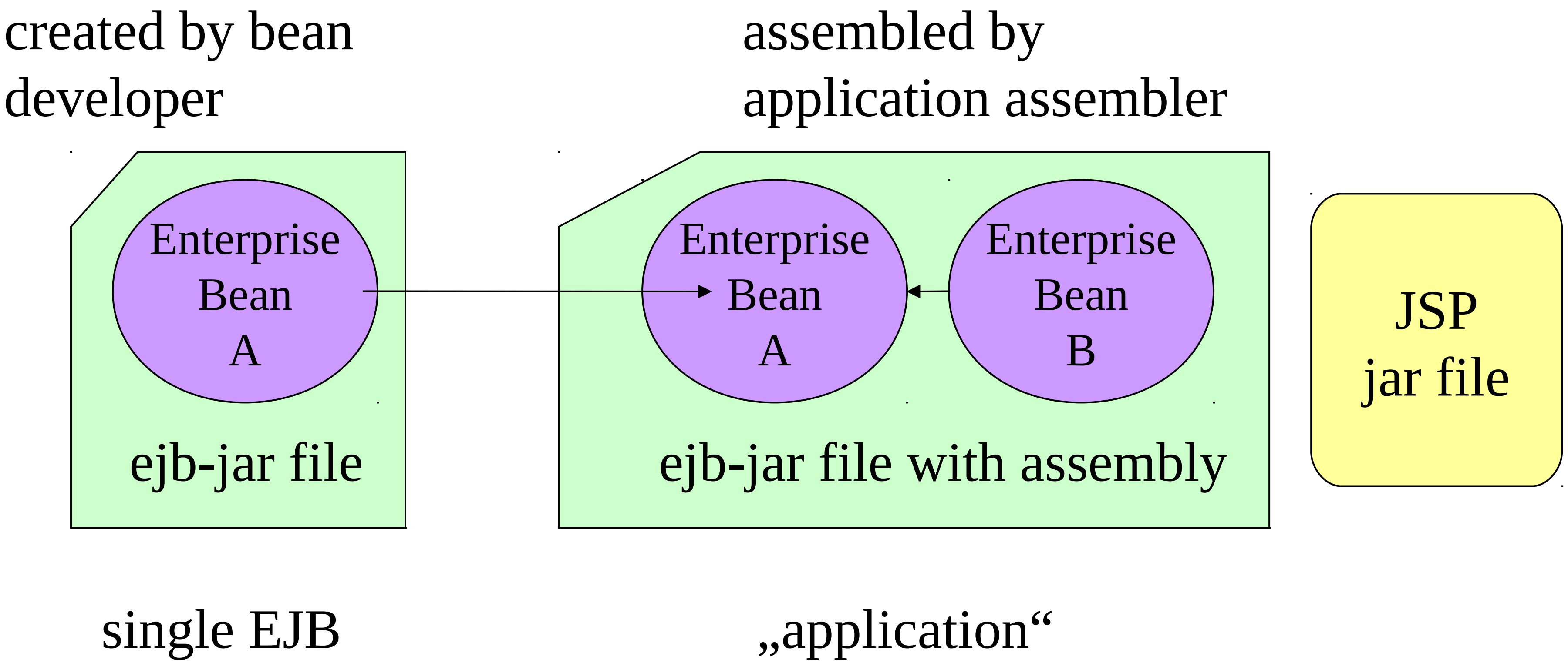
from an interview with Ted Neward, theserverside.com. He also wrote „Effective Enterprise Java“ (resources)

Component Example: Enterprise Java Beans



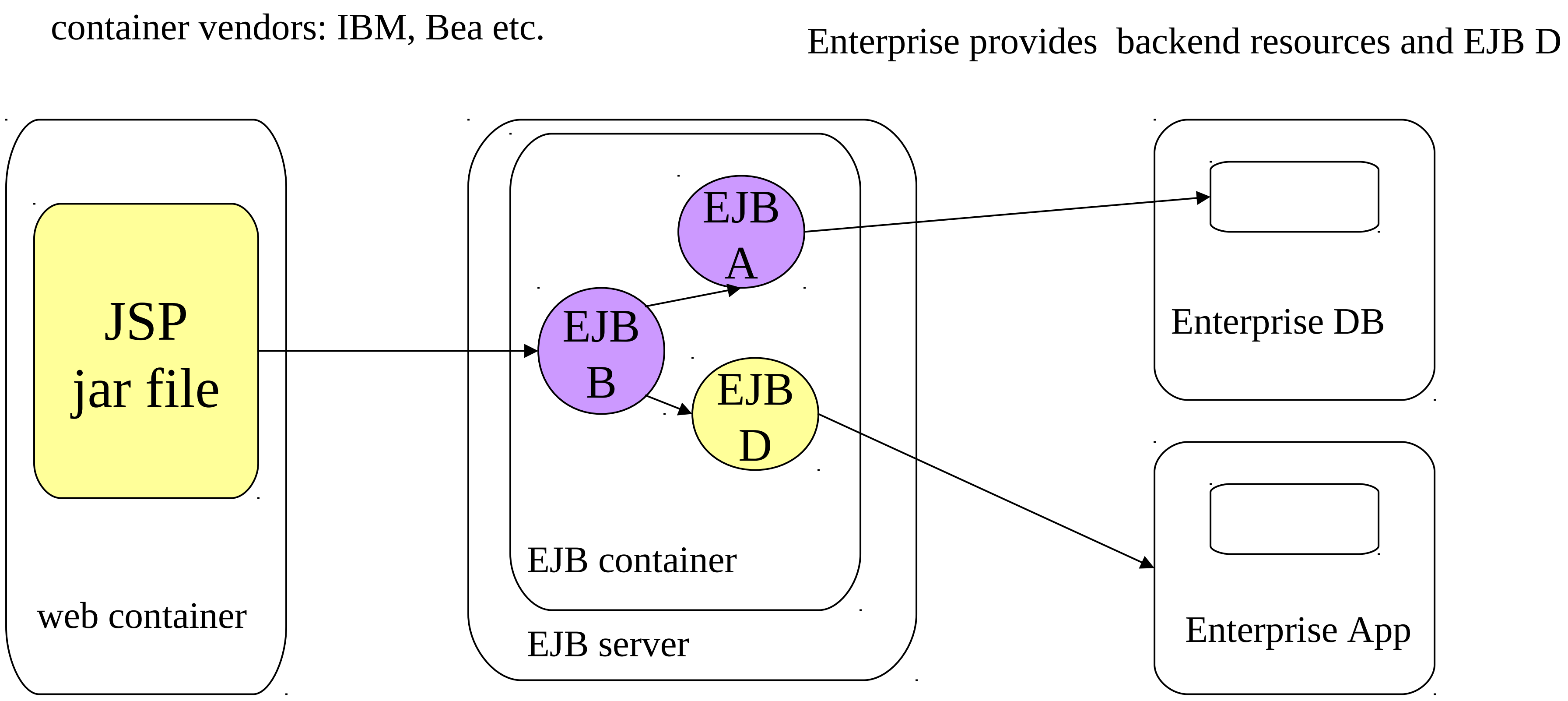
Early versions of EJB did a fine grained mapping between business entities to beans to DB-tables. All components („enterprise beans“) were remote. Newer versions have local interfaces as well and session beans can compose a number of beans into a component interface. Support for business processes (workflow) is weak but will probably improved through „activity beans“ etc.

EJB roles and parts (1)



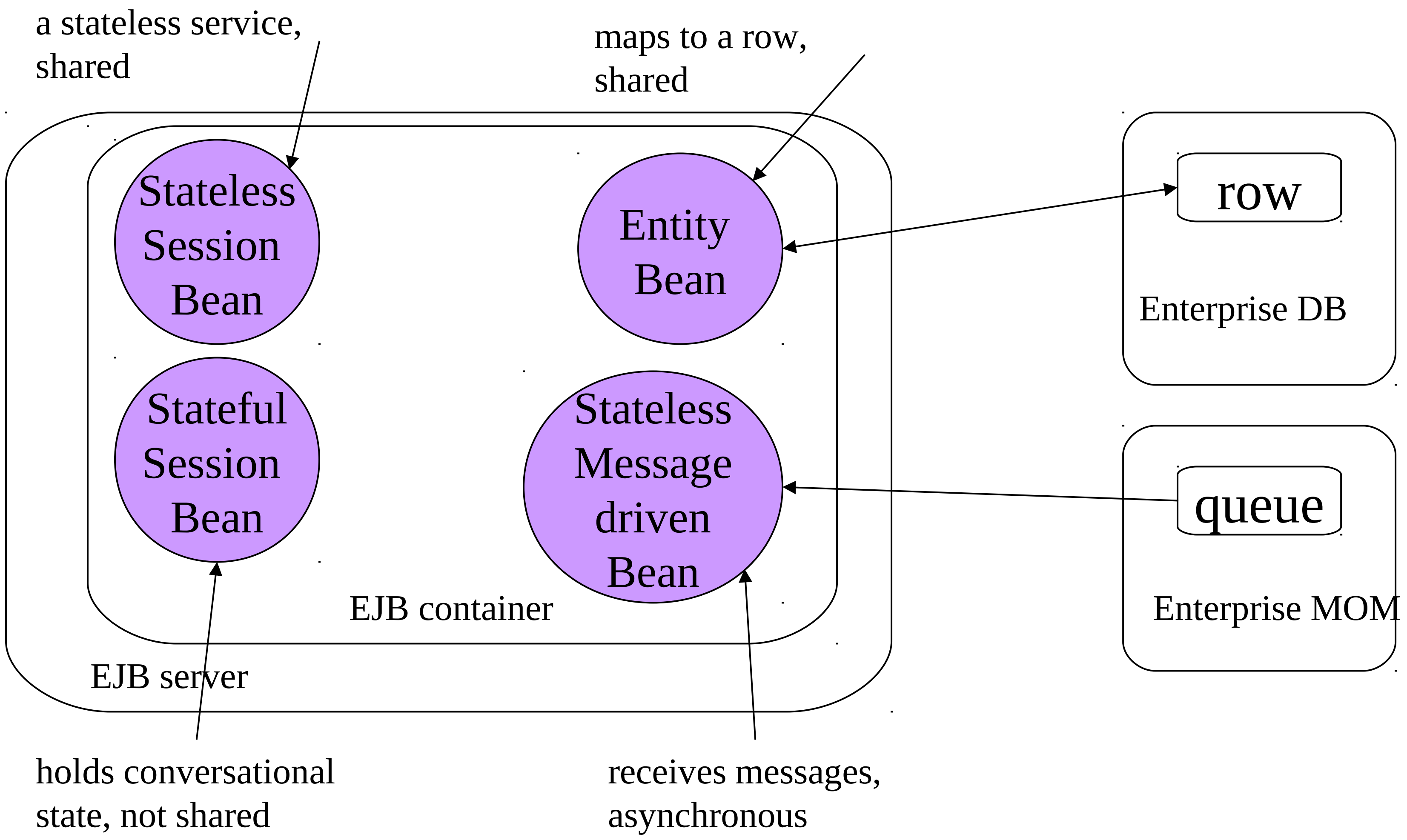
bean developer and application assembler are different roles. Assembly happens through integration of EJBs into bigger ejb-jar files and adjusting the meta-information (deployment descriptor). The „application“ can contain non-EJB parts like JSPs or servlets. EJB B uses EJB A.

EJB roles and parts (2)



The application is now deployed into a web container and an EJB container. EJB server and container are usually from the same vendor. The EJBs are connected to the enterprise backend resources (DB or applications) by the deployer of the application or system management. The EJB B is connected to an enterprise internally developed EJB D.

EJB Component Model



Four different EJB types allow for scalability (stateless services), client code on server side (conversational state), asynchronous processing and the representation of company data (entities). Entity Beans are permanent while stateful session beans do not survive a server crash.

Session Beans (stateful and stateless)

- Are per client (un-shared), except stateful session objects which represent client code moved to the server.
- Can participate in transactions (if session-synchronization interface is used)
- may access databases but do not directly represent persistent objects
- short-lived
- removed when container crashes

session objects, especially stateless ones scale best. EJB servers should be able to support large numbers of those objects

Entity Beans (deprecated since 3.0)

- Are always shared and therefor protected through transactions.
- Their lifetime exceeds the server lifetime
- represent important company data
- can be persisted through container mechanism (Container managed persistence, CMP)
- can do their own persistence (Bean managed persistence, BMP)

entity objects have a unique identity which is visible to clients (primary key). A client can request a „handle“ which is a persistent pointer to an entity object which allows the client to contact the object even after a long time. In 3.0 persistence is no longer a concern for EJB. It is now covered in the Java Persistence API!

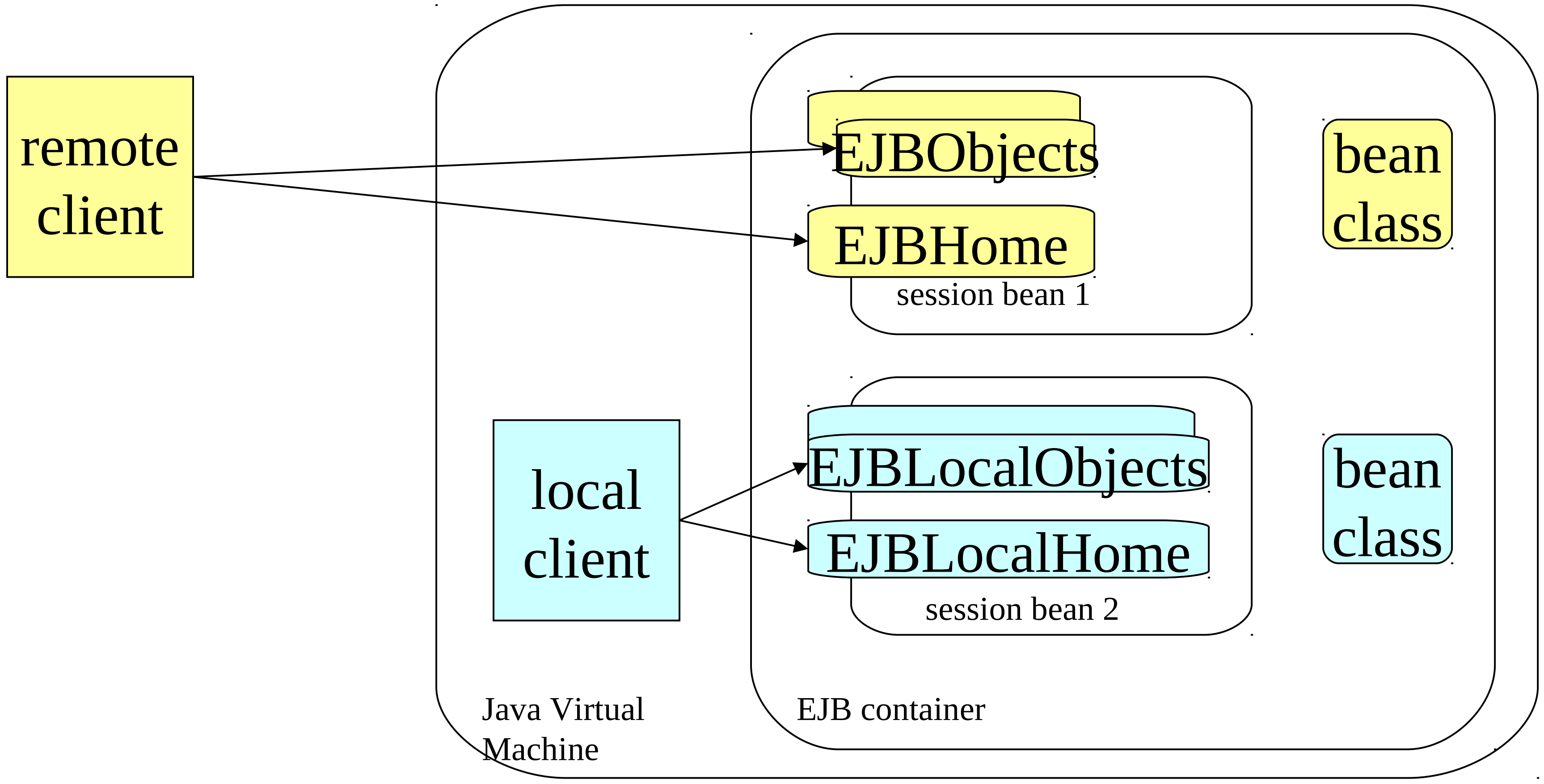
Message Driven Beans

- are invoked asynchronously
- no client context available during processing
- can be transaction aware
- short-lived and stateless, removed when container crashes
- Do not map to company data directly

Transaction aware means that the message receipt and processing can be enclosed in one transaction. If the bean crashes during processing the message is counted „un-read“ and will not be lost. It can be processed after re-start of the container.

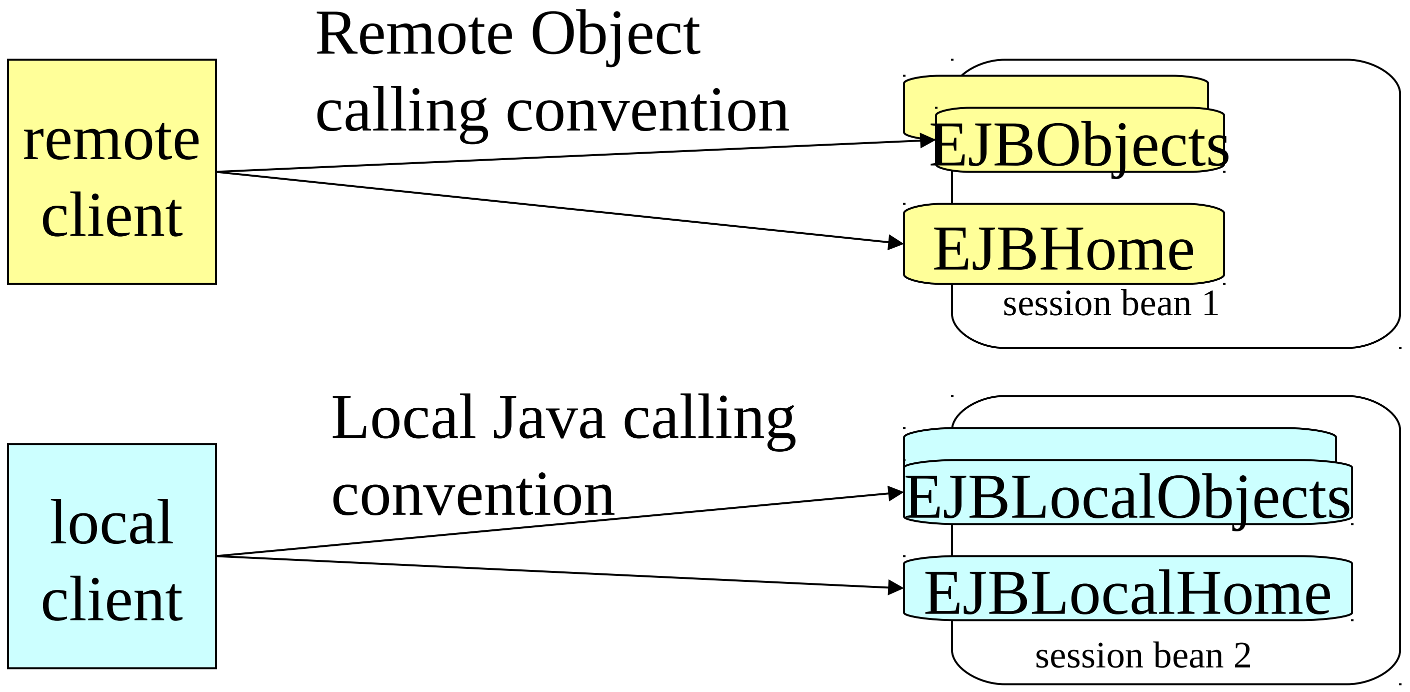
The message driven beans have been integrated into the general EJB framework to re-use the EJB container services (transactions, security, concurrency, deployment description etc.)

Client View of EJBs



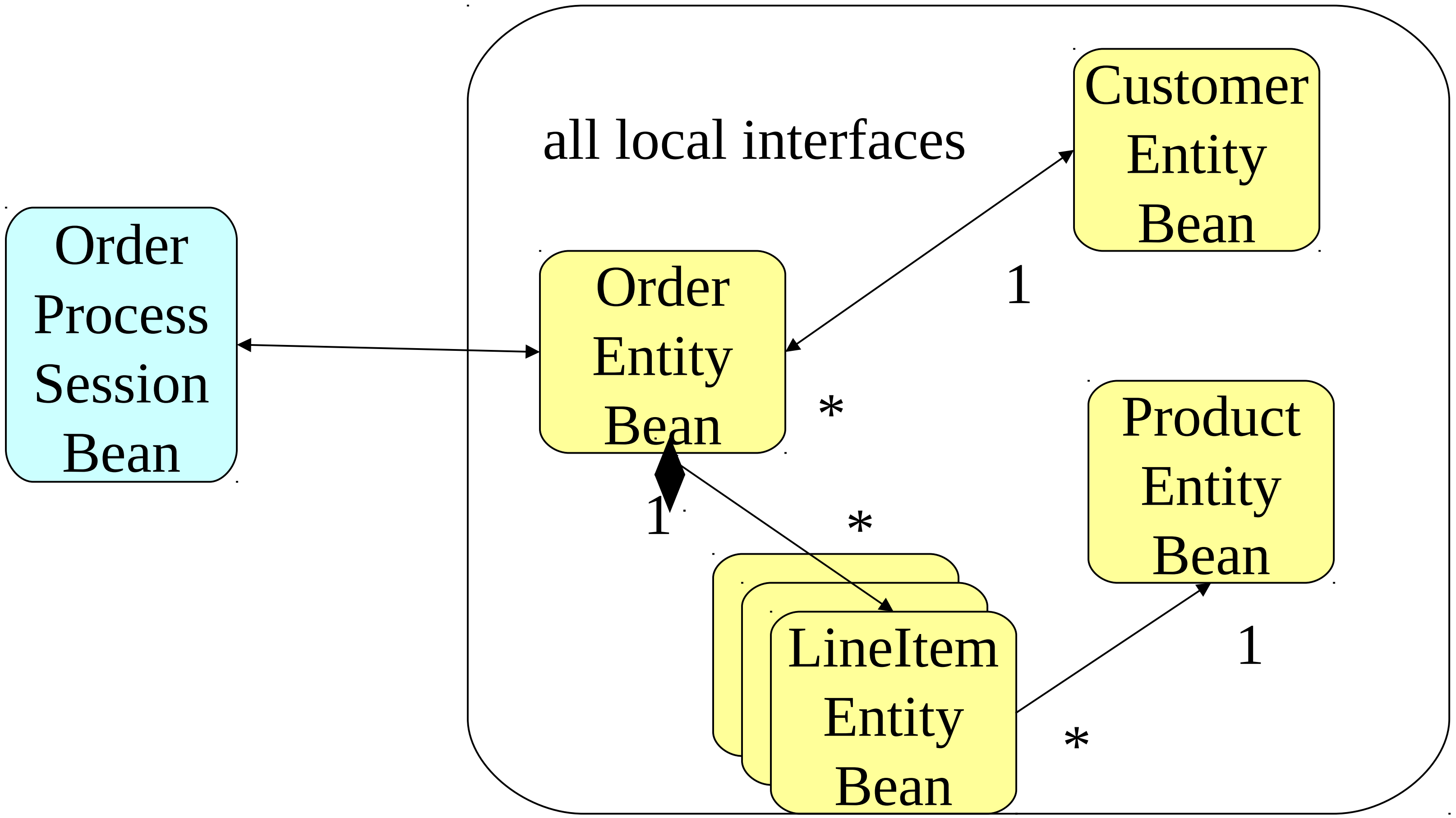
a client NEVER accesses directly the bean class. A EJB can offer a remote and/or a local interface. Clients of the local interface need to be in the same Java virtual machine as the bean container. The home contains mostly lifecycle methods while the EJBObject deals with identity and handles. The business logic is contained in the bean class.

Local vs. Remote Interfaces



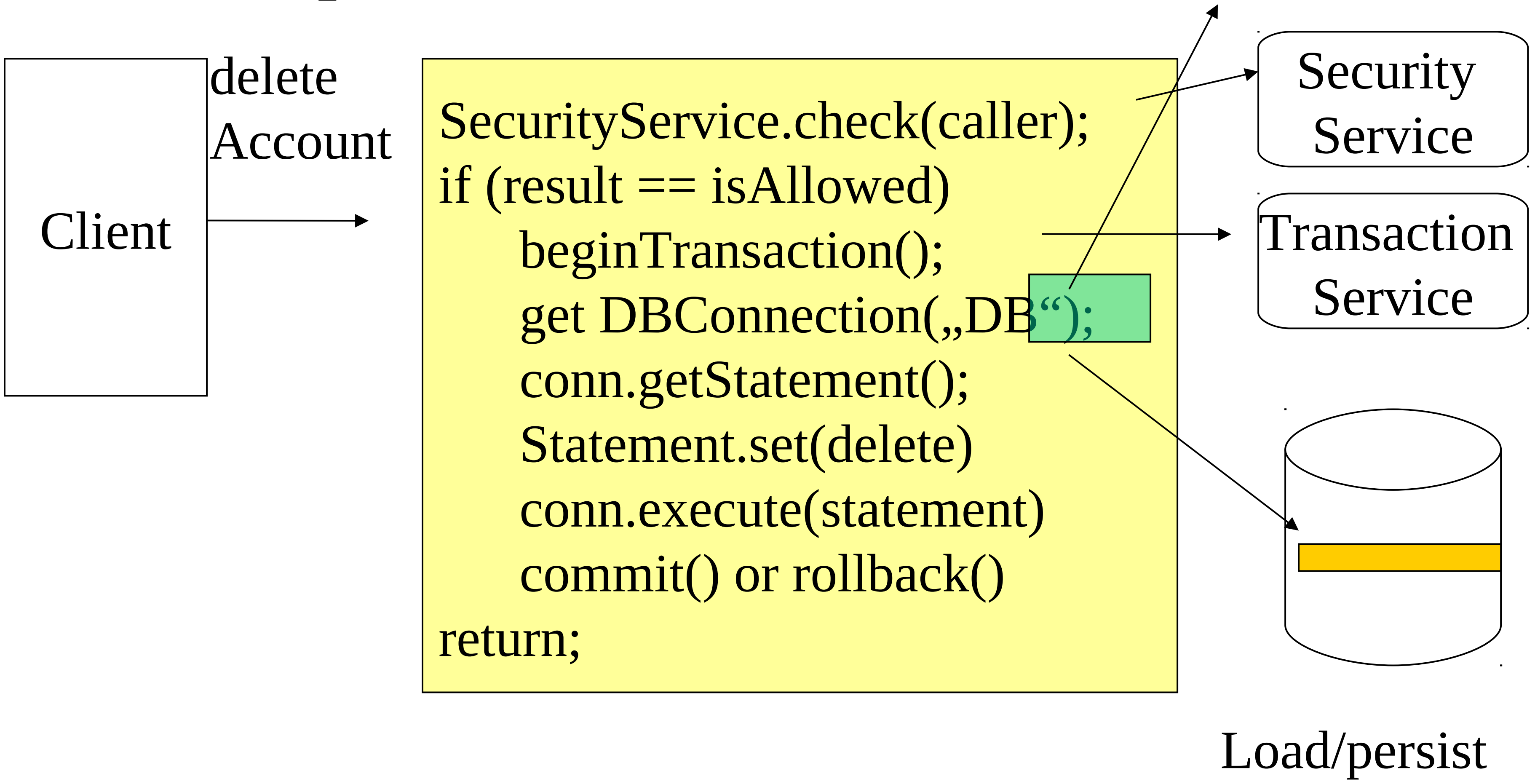
Calling the local interface of an EJB uses the same semantics as a local java call: Value objects are moved BY REFERENCE, meaning client and EJB will SHARE objects. The remote calling semantics will of course require that value objects are copied. Bean implementers who want to provide both interfaces must respect the different calling conventions.

Local Interfaces and persistent relationships



The introduction of local interfaces allowed container managed relationships. If e.g. an order entity is deleted, the container will adjust references and delete aggregated objects like LineItem. Objects involved in one-to-one relationships are effectively moved if they are added to another objects relationships. Not so for 1toMany or Many-to-many relationships.

No Separation of Concerns and Context



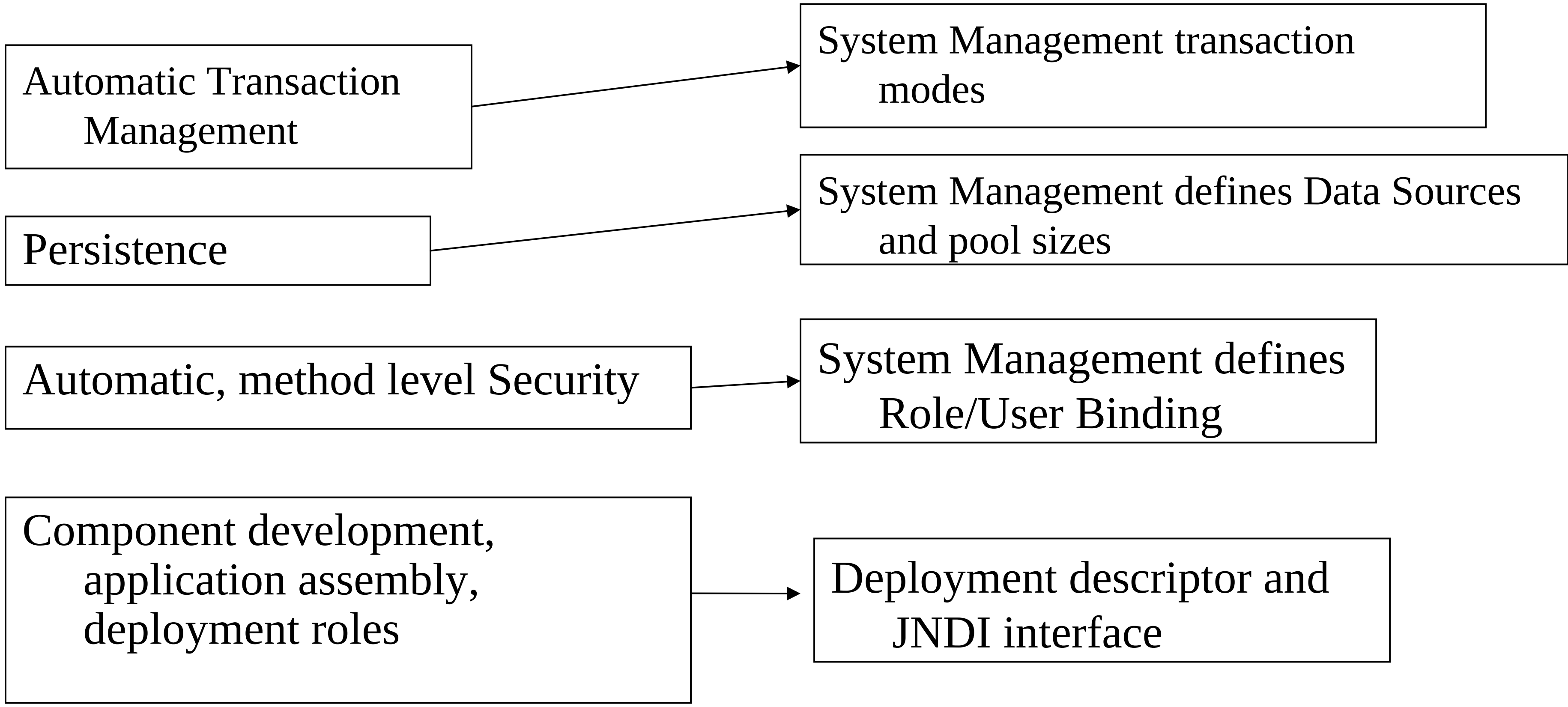
```
SecurityService.check(caller);
if (result == isAllowed)
      beginTransaction();
      get DBConnection(„DB“);
      conn.getStatement();
      Statement.set(delete)
      conn.execute(statement)
      commit() or rollback()
return;
```

A lot of code to perform just a tiny business function! Programmers need to know several service APIs and how they work. AND: Information about the context of the system (Database names, user roles etc. is embedded in the application.

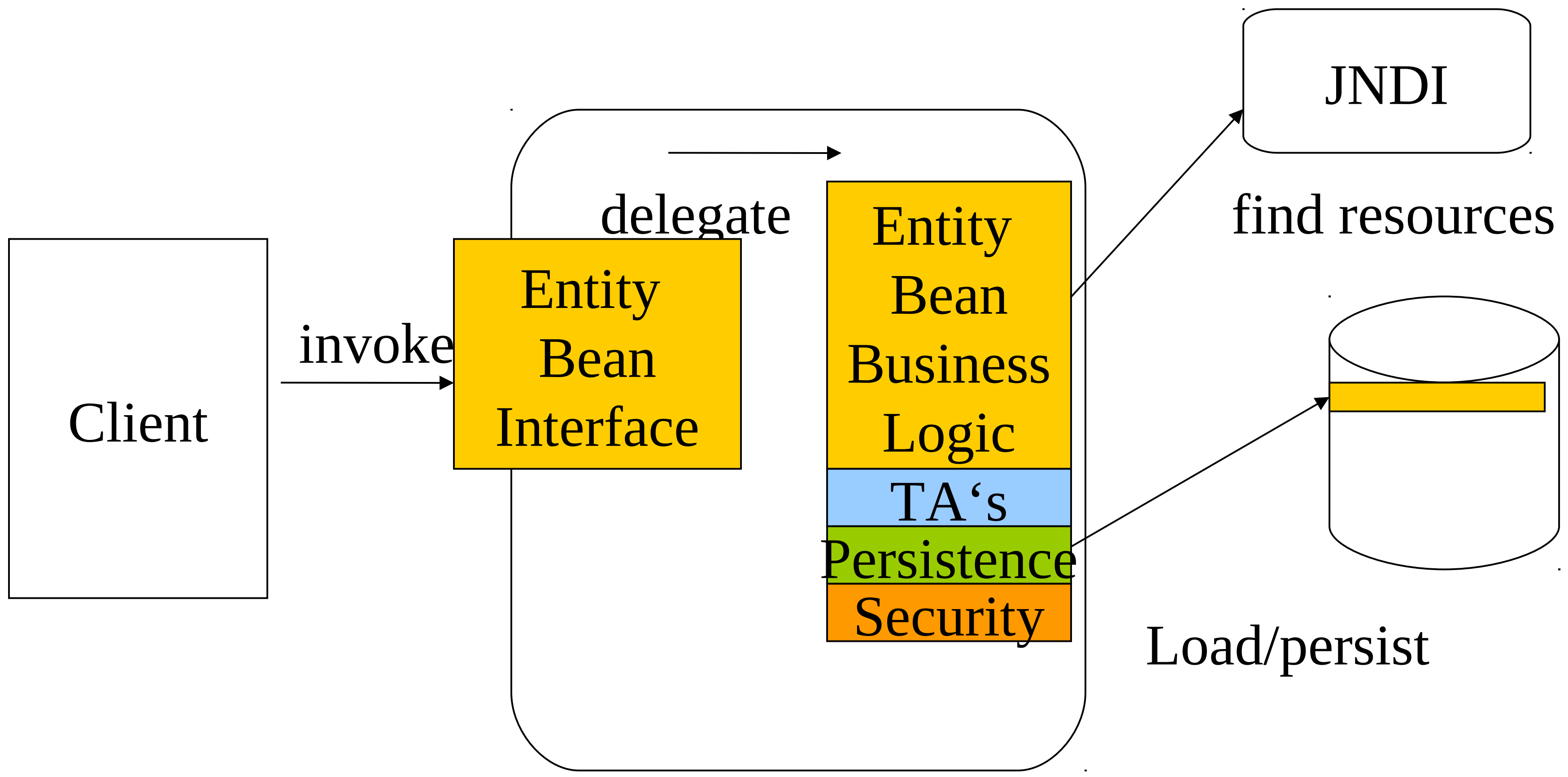
Separation of concerns and context

EJB Framework (Separation of **concerns**):

Deployment (Separation of **context**):

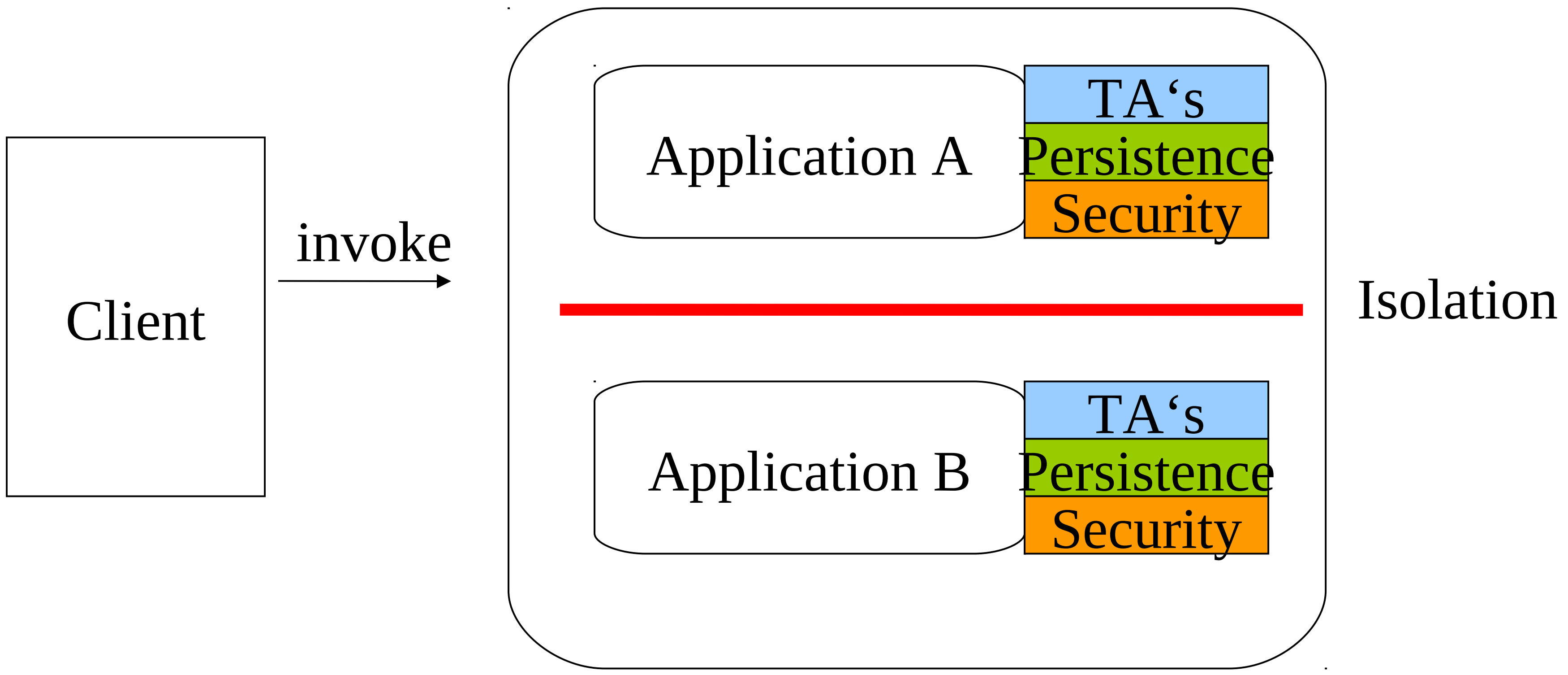

Automatic Transaction Management
System Management transaction modes
Persistence
System Management defines Data Sources and pool sizes
Automatic, method level Security
System Management defines Role/User Binding
Component development, application assembly, deployment roles
Deployment descriptor and JNDI interface

EJB Container



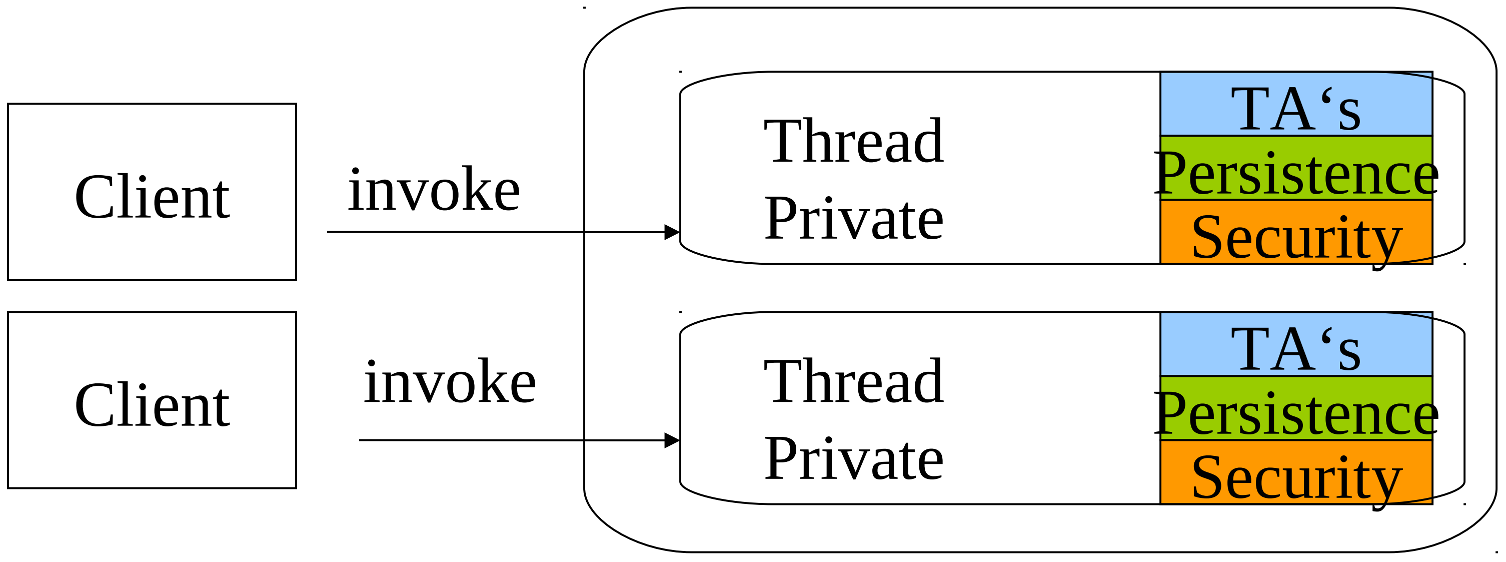
At the point of interception the container provides the following services to the bean:
Resource management, life-cycle, state-management, transactions, security

More Container Concerns



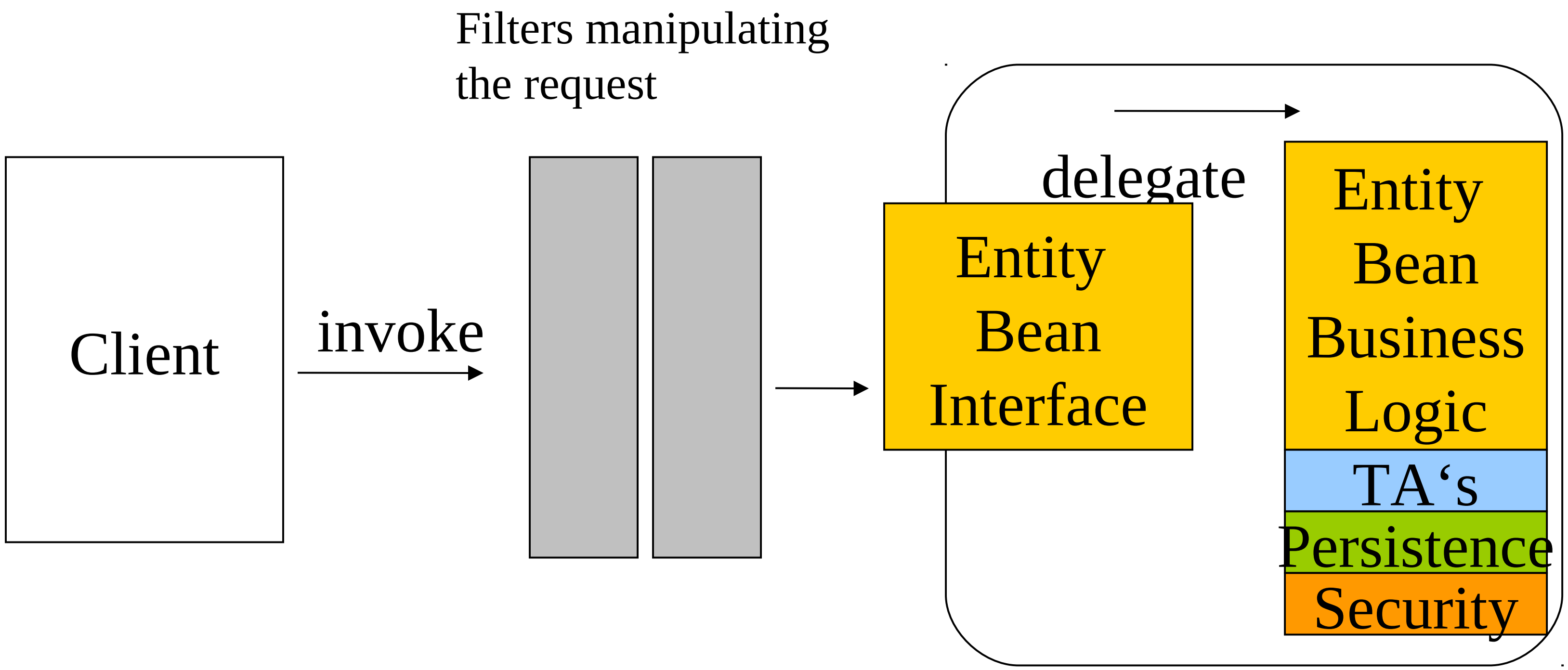
Containers more and more take over roles from operating systems. A key role is to isolate different applications from each other. Currently J2EE uses class loaders for this purpose. A better concept that does not mix loading with isolation is needed.

Containers and Threads



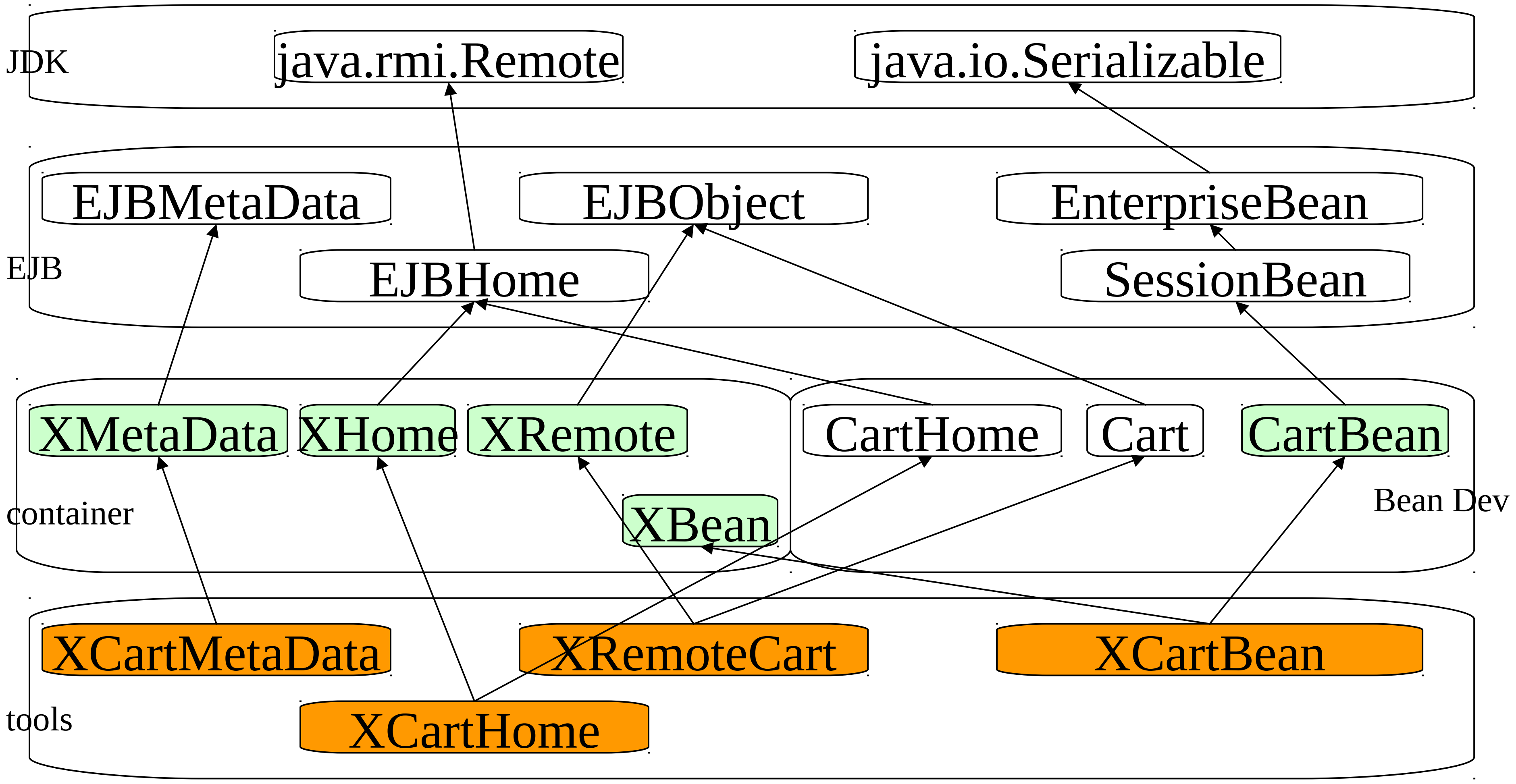
A container manages resources across applications. It stores context and session information in threadlocal storage. This is the reason why container managed applications are not allowed to create their own threads. These threads would not have the proper meta-data and context information. EJB3.0 offers a managed service for connectors to create threads. Applications should not assume resource management so that the container can chose the proper policy.

Interceptors – the missing link?



As Ted Neward points out most remoting technologies offer interceptors/filters to allow applications to manipulate the requests. Servlet filters, CORBA interceptors or RMI SocketFactories all follow the interceptor pattern and can be used e.g. to establish session handling or enforce security checks. Just EJB does not expose the mechanism to applications.

Classes and Interfaces in EJB 2.1



The orange classes mix business logic (CartBean) with framework behavior (Home, Object etc.). Note that CartBean does NOT derive from Cart Interface. White boxes are interfaces only.

Entity Bean - Container Contract (1)

- setEntityContext(EntityContext) Bean stores context as an interface to the environment. Usage depends on state.
- PrimaryKeyClass ejbCreate<Method>: Actions related to bean instance construction.
- ejbPostCreate(): Bean identity is now available
- ejbActivate(): Bean can acquire necessary resources.
- ejbPassivate(): bean releases resources, expecting to be put back into the pool.
- ejbRemove(): last chance for the bean before destruction.
- ejbstore(): bean should update ist internal state, expecting it to be synchronized with the DB right after this.
- ejbload(): bean should update ist internal state, expecting that ist virtual fields have just been read from the DB.
- ejbFind(), ejbSelect(). Query methods generated at deployment
- ejbHome<method>: business logic that does not require an object identity

Entity Bean - Container Contract (2)

what can be done in the framework methods depends on:

-a transactional context available?

-an object identity available?

-a local or remote view available?

-a client security context available?

See the EJB spec. for full details on both bean provider and container vendor responsibilities.

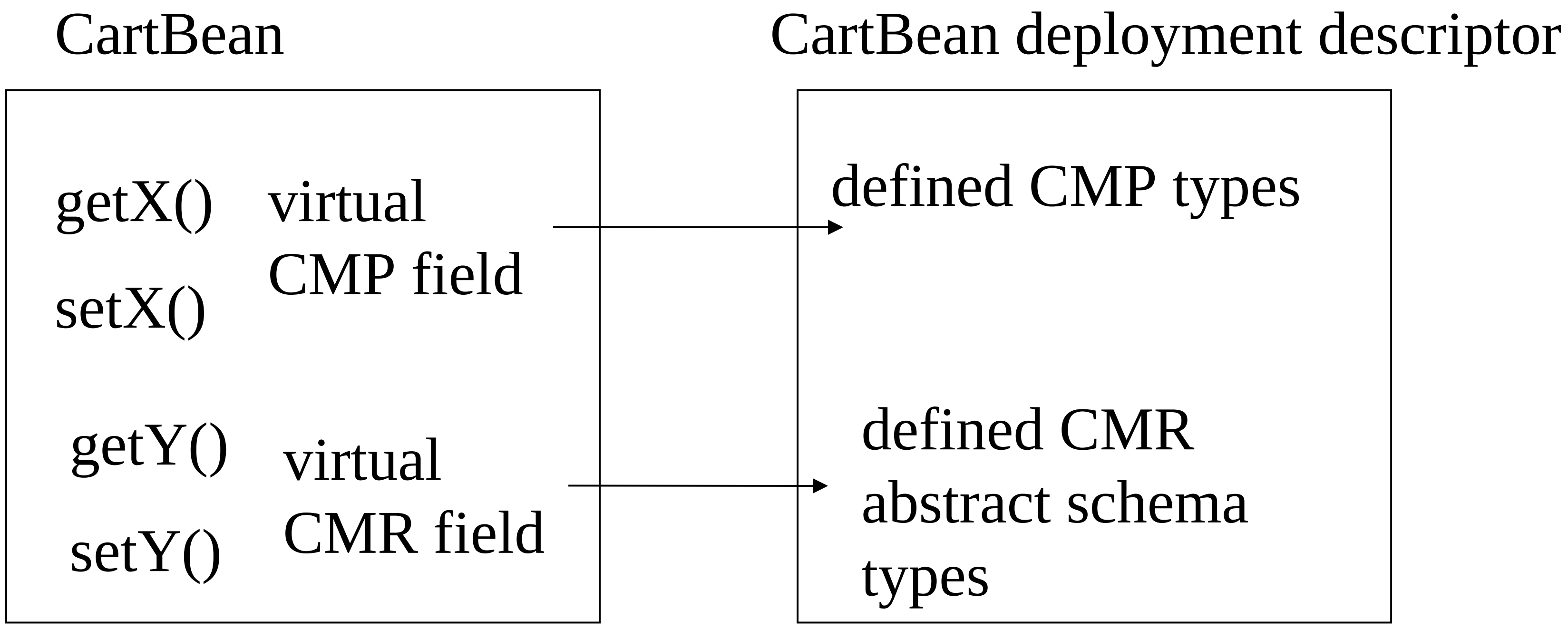
Bean Managed vs. Container Managed Persistence

Bean needs to perform ist own persistence. When this happens is controlled by container

Bean state is completely stored and loaded by container.

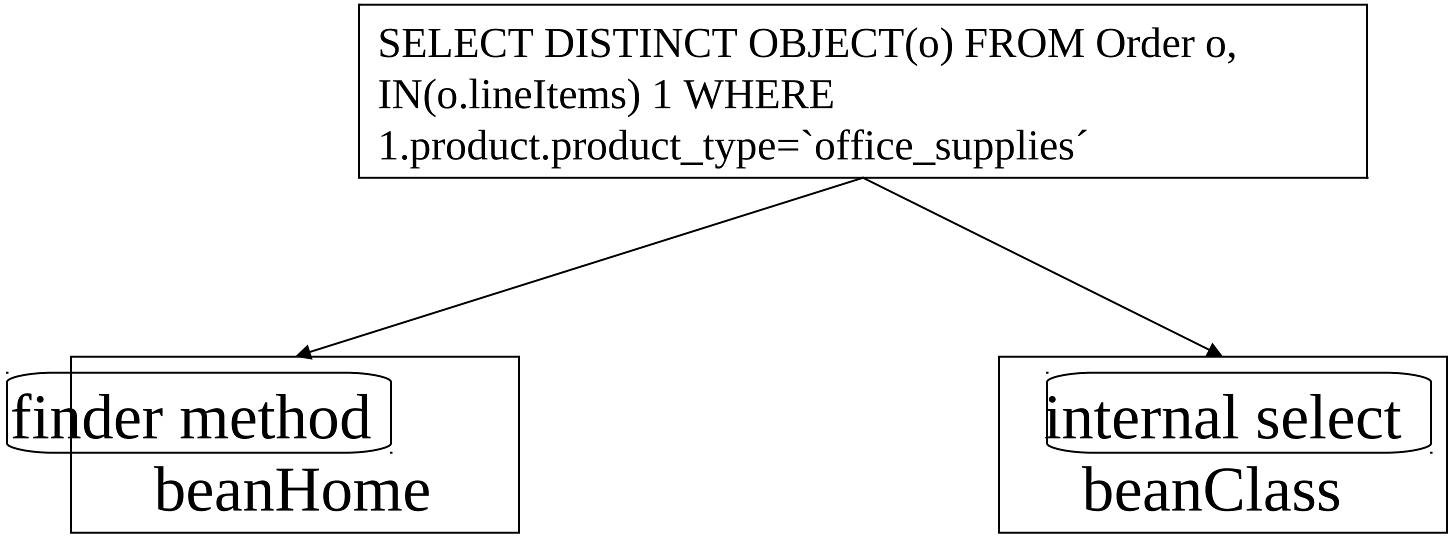
Container managed persistence is clearly the way to go in the future. Bean managed persistence is not portable and requires adjustments to different datastores etc.

EJB 2.1 Virtual Fields and Abstract Schema Types



Previous EJB Releases did use regular field definitions in the beans class. Rel.2.0 lets the bean developer only define getters and setters, no fields. The deployment descriptor maps getters/setters to types. Advantage: container can now use lazy load techniques because the bean cannot access persistent fields directly – only through the getters and setters.

EJB 2.1 Query Language



SELECT DISTINCT OBJECT(o) FROM Order o, IN(o.lineItems) 1 WHERE 1.product.product_type=`office_supplies´
finder method
beanHome
internal select
beanClass

EJB QL describes queries in terms of abstract schema types and ejb_names etc. in the deployment descriptor. This allows query processors to optimize queries by mapping them to the real datastore query language. Otherwise queries would be dependent on a specific datastore.

EJ Beans Environment: JNDI naming context

Initial Context:

java:comp/env......

java:comp/env/foo...env.entries

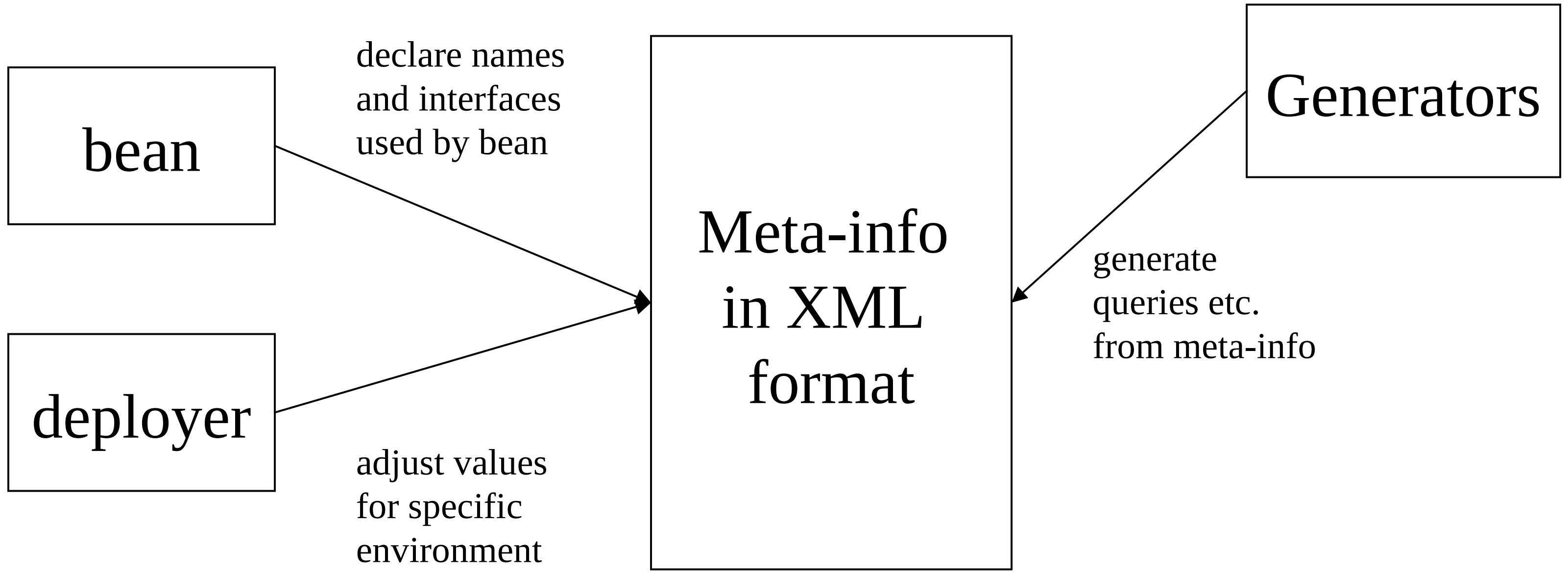
java:comp/env/ejb.. EJB beans

java:comp/env/[jdbc|jms]
Resource Manager Connection Factories

java:comp/env/jms/
StockqueueResource
environment references

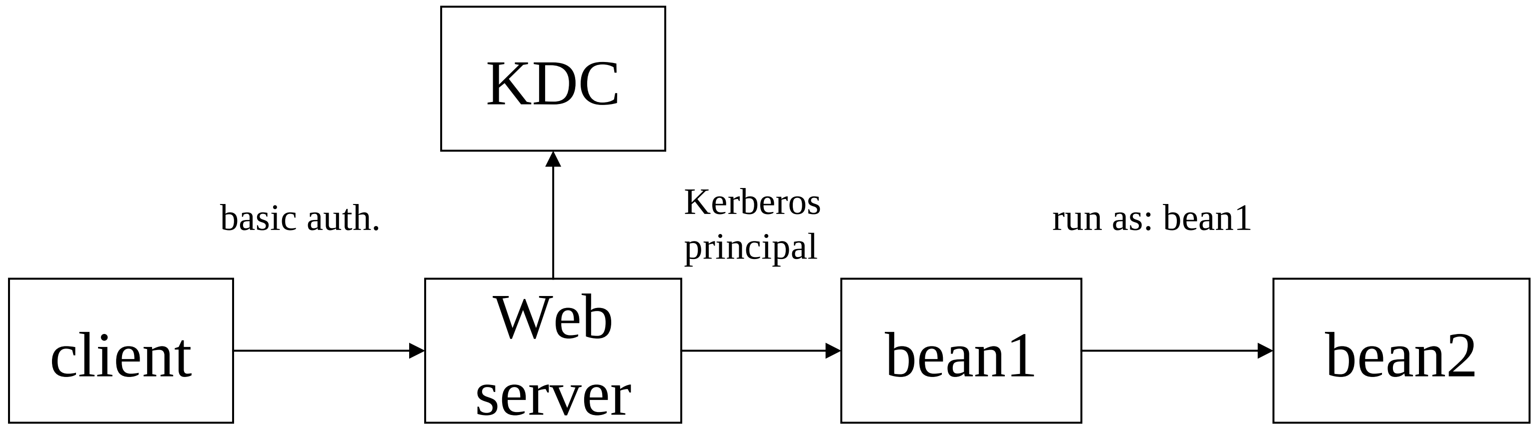
Beans locate all their resources through JNDI calls, allowing deployers to place proper services there. All lookups can be manipulated via the deployment descriptor.

Deployment Descriptor


bean
deployer
declare names and interfaces used by bean
adjust values for specific environment
Meta-info in XML format
Generators
generate queries etc. from meta-info

Interfaces create black box views on components. The deployment descriptor is the main data structure that exposes internals in a controlled way and transports meta-information across working steps. This lets. e.g. the bean developer specify security roles without knowledge of real security roles in a target environment. The roles can be mapped during deployment

Security Support



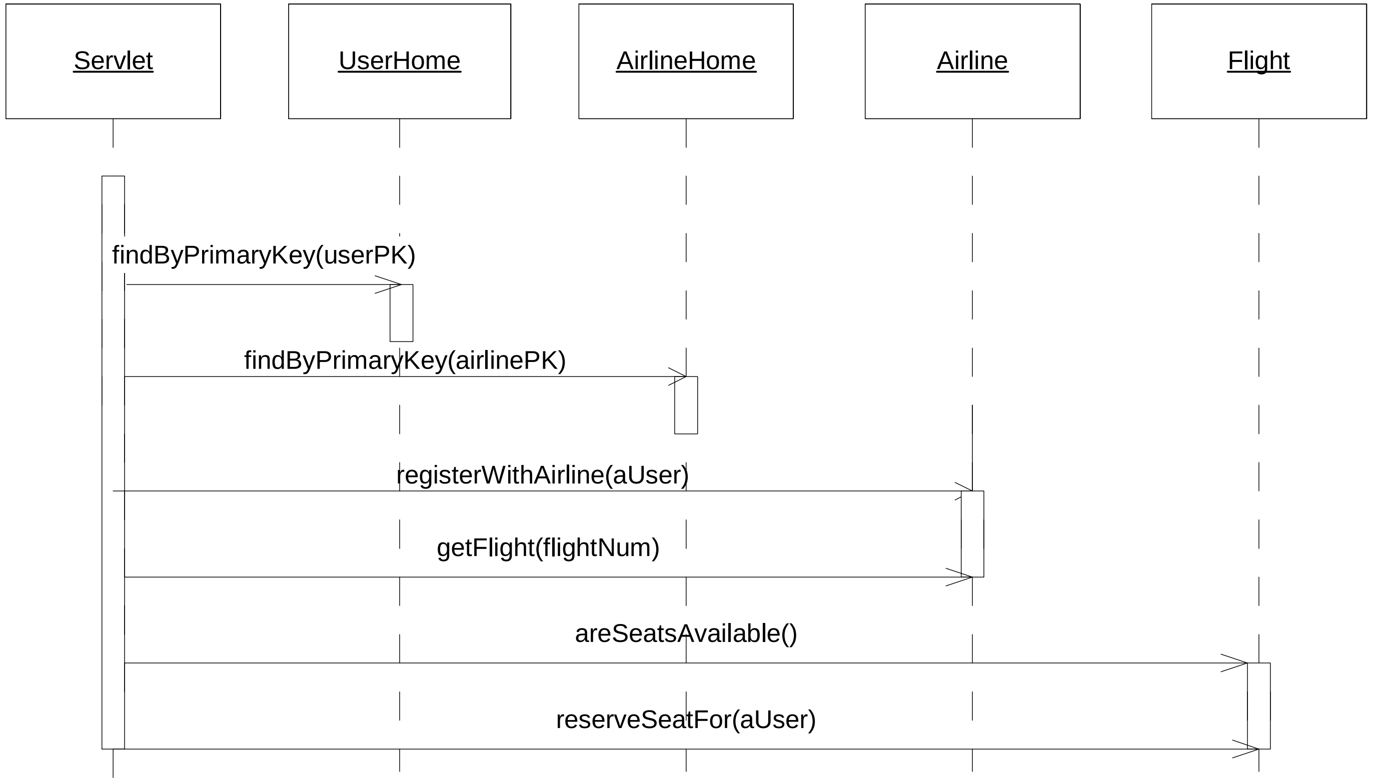
EJBs allow two security calling modes: principal delegation or „run as“ identity. Even if „run as“ is specified in the Depl. Descr, a getCallerPrincipal() at bean2 will return the original caller (client)

Transaction Modes

- Not supported
- Required
- Supports
- RequiresNew
- Mandatory
- Never

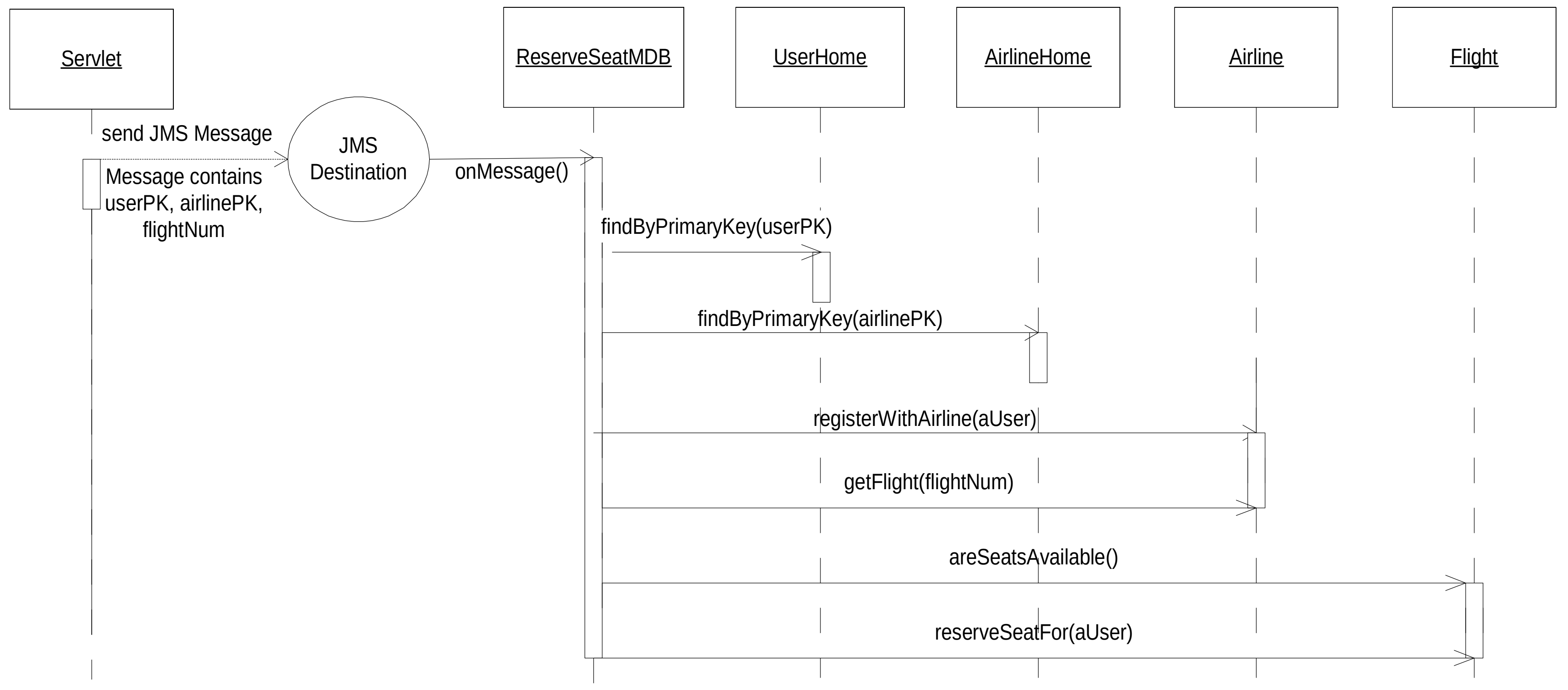
the transaction modes are specified in the deployment descriptor. Depending on the modes, TA‘s are either created, taken over from caller or exceptions are thrown.

EJB Anti-Pattern



this use of EJBs will result in poor performance and user response time (synchronous calls) and lack of transaction consistency (single calls). Also, the business logic is implemented in the servlet (client) area. Other bad design issues: tight coupling of client to bean interfaces.

EJB Pattern: Message Facade



A „facade" contains now the business logic and performs the calls to participating beans. Since the client does not need an immediate response, a message bean is ideal. Transactional consistency is guaranteed even if server crashes. The message won't get lost. A session bean fassade would have been possible as well but suffers from the synchronous calls to beans. (from EJB Design Patterns)

EJB 2.1 Best Practises

- Use JDBC for read-only data (lists etc.)
- Use generic access container to transport data across containers
- Use portable primary key generator
- Use facades to encapsulate business logic and separate development teams
- Use Singletons correctly (non-blocking)
- Implement own auto key generator

With every new technology finding the best practises takes quite some time. They make the difference between projects that fail and those that fly. (www.theserverside.com EJB Design patterns)

EJB 2.1 shortcomings

- Large number of artifacts for the programmer to control
- Meta-data separated in deployment descriptor instead of code
- Home interfaces and finding of remote objects tedious
- Performance problems in the O/R mapping due to abstract schema approach
- No rapid prototyping possible
- Entity beans overloaded with security, transactions and persistence.

Session facade objects typicalle handle security and transactions. But the highly artificial O/R mapping layer of EJBs may not be enough to justify the effort.

New ways for meta-data

```
/** * This is the EJB Receiver Xbean * *
 @ejb:bean type="Stateless" * name="ejbReceiver" *
                  jndi-name="org.xbeans.ejb.receiver.Receiver" *
                 display-name="EJB Receiver Xbean" * * ... other javadoc tags ... */
  public class ReceiverBean implements SessionBean, DOMSource { …
```

The XDoclet source code annotation system brought meta-data back into Java sourcecode. With XDoclet it was much easier to generate EJB artifacts. C# supports source code annotation as well. The whole concept is so successful that it became part of Java 5.0 (annotation system) where developers can create annotations for tool-time, compile-time or runtime.The Java reflection API was extended to let objects read those annotations.

New ways for O/R mapping

- regular java objects (POJOs) are either annotated (hibernate) or byte-code modified (JDO) to become persistent objects
- SQL is back: The trend towards generic, abstract schema mappings seems to be over

Developers always had problems with the highly abstract was EJB entity beans where mapped to persistent stores. JDO and others proved that plain old java objects should be all that developers need for persistence.

Overview of EJB 3.0 features

- More descriptive power: instead of marker interfaces developers can use java annotations to express what kind of behavior a container should provide. No more deployment descriptors needed.
- Home interfaces and ejb_create() method gone.Intitialization now left to the client
- Entity beans are originally only beans and become attached to a persistence layer by association with an entity manager
- Business interface can be automatically created from the developers only bean class.
- Mapping of entity beans is now directly to a DB table and rows/columns. The names of those can be defined in code using annotations
- Direct SQL support.
- Query language now close to SQL
- Source code annotation used to find references, define relationships and queries

Warning: embedding SQL directly can severely limit the portability of your components. See: Anil Sharma (resources). The specification learned a lot from e.g. Springs dependency injection.

Overview of EJB 3.1 features

- Local view without interface (No-interface view)
- .war packaging of EJB components
- EJB Lite: definition of a subset of EJB
- Portable EJB Global JNDI Names
- Singletons (Singleton Session Beans)
- Application Initialization and Shutdown Events
- EJB Timer Service Enhancements
- Simple Asynchrony (@Asynchronous for session beans)

@asynchronous example

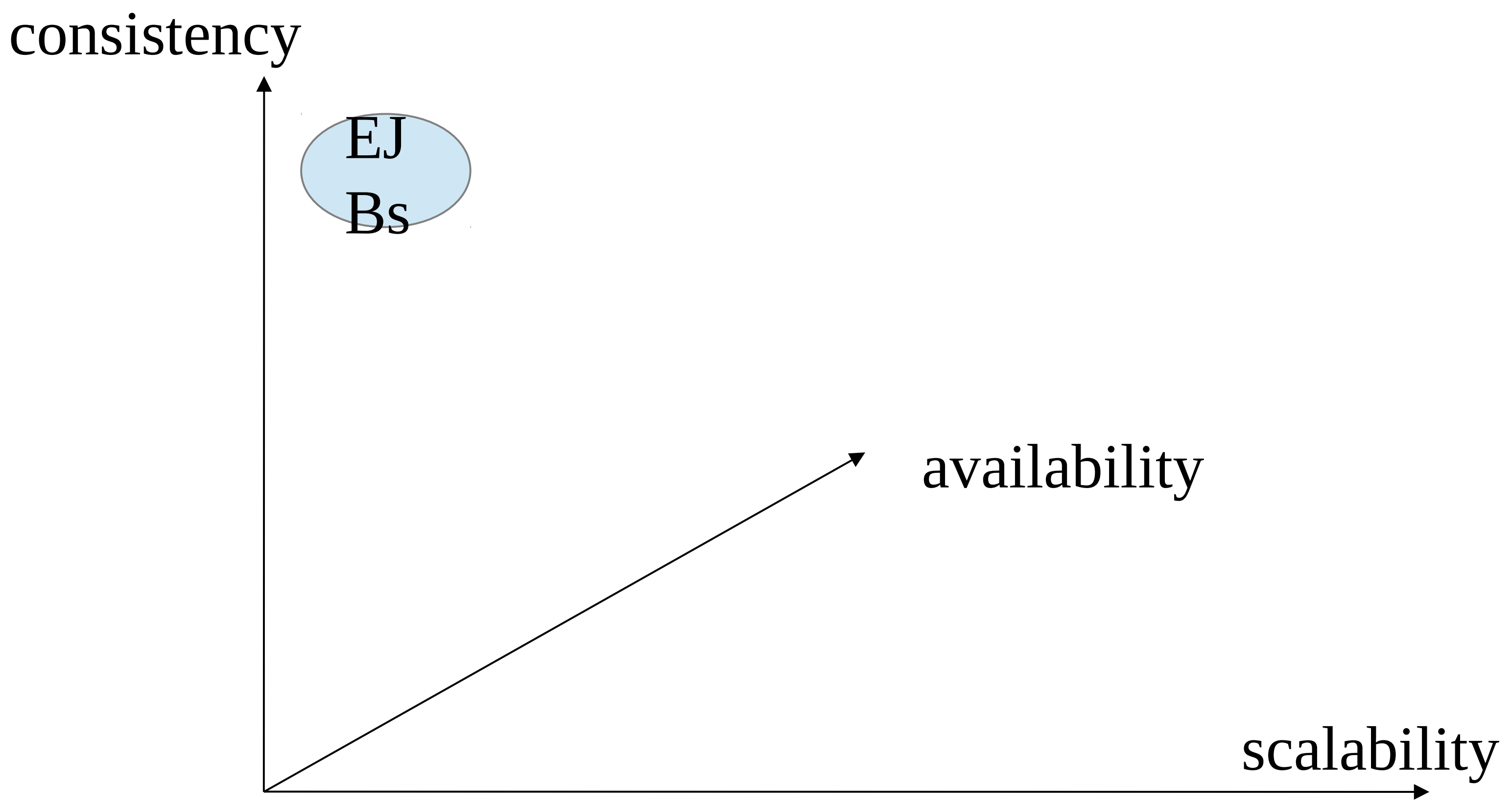
```
@Stateless
@Remote(HelloEjbAsynchronousRemote.class)
public class HelloEjbAsynchronous implements HelloEjbAsynchronousRemote {
    @Asynchronous,   @Override
    public Future<String> ejbAsynchronousSayHello(String name){
        System.out.println(new Date().toString()+" - Begin - HelloEjbAsynchronos-
      >ejbAsynchronousSaySayHello "+name);
        try{
          Thread.sleep(5*1000);
        }catch (Exception e){
           e.printStackTrace();
        }
        System.out.println(new Date().toString()+" - End - HelloEjbAsynchronos-
      >ejbAsynchronousSayHello "+name);
        return new AsyncResult<String>("Hello "+name);      } }
```

Example from: Patrick Champion, http://paddyweblog.blogspot.com/2010/04/ejb-31-asynchronous-session-beans.html

Lessons Learned

- It takes many years to define a complex framework for components and to make it scale.
- It takes many iterations to get the interfaces right so they can be implemented in a way that performs.
- Developers facing abstractions often do not understand the consequences.
- Frameworks sometimes force developers to do code duplication (e.g. finding objects from JNDI). This is tedious (better: dependency injection with spring).
- Don't couple too many concerns in one framework (Transactions, persistence, security)
- Code generation is nice but requires tooling. Tooling needs customization to work.
- Don't apply new technology on a large scale before “best practice patterns” exist. Complex technology needs those patterns.

Dimensions of Distributed Systems



Coming up next: CAP, CALM, CRDTs and how I learned to love copying data ….

Resources (1)

- Peter Herzum, Oliver Sims, Business Component Factory. From the „father of business components – Sims“ the book on how to build enterprise wide distributed business component solutions.
- DeMichiel et.al., Enterprise Java Beans Specification Version 2.0, Sun Microsystems 2001. Has gotten a little bit bloated over the years, still easy to read.
- Mastering Enterprise Java Beans II, www.theserverside.com, if specs aren‘t your thing yet.
- EJB Design Patterns, www.theserverside.com , very important „best practises“ for EJB programming.
- The CORBA component model, www.omg.org

Resources (2)

- Anil Sharma, EJB 3.0 in a Nutshell, http://www.javaworld.com/javaworld/jw-08-2004/jw-0809-ejb_p.html Describes the changes in EJB 3.0 on a few pages.
- Dion Almaer, Using XDoclet: Developing EJBs with Just the Bean Class http://www.onjava.com/pub/a/onjava/2002/01/30/xdoclet.html
- Nicolas Schmid, Introduction to XDoclet. http://www.kriha.de/krihaorg/dload/uni/generativecomputing/generation/XDoclet.pdf
- Ted Neward, Effective Enterprise Beans. Like his book on server side Java Ted tackles the real problems of development: Class loading, performance, scalability. If you really want to understand EJBs get this book.