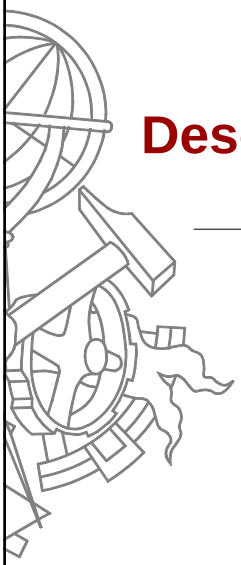


Ambientes de Desenvolvimento Avançados

http://www.dei.isep.ipp.pt/~jtavares/ADAV/ADAV.htm

Aula 17

Engenharia Informática

2005/2006

José António Tavares
jrt@isep.ipp.pt

Agenda

Introduction to EJB

Java 2 Platform, Enterprise Edition (J2EE)
http://java.sun.com/j2ee/index.jsp

Componentes e a Tecnologia JAVA

Unidade de distibuíção binária	Ficheiro JAR
Fornecimento de serviços	Classes
Contrato explicito	Interfaces
Reutilização	Referência ao ficheiro JAR no CLASSPATH
Composição por terceiros	Composição e especialização (herança) de classes

Enterprise JavaBeans

The Enterprise JavaBeans architecture is a component architecture for the development and deployment of component-based distributed business applications. Applications written using the Enterprise JavaBeans architecture are scalable, transactional, and multi-user secure. These applications may be written once, and then deployed on any server platform that supports the Enterprise JavaBeans specification.

… and it is Java based

(Sun Microsystems' definition)

JavaBeans

- JavaBeans define a *software component model* for Java
  - third party ISVs can create and ship Java components that can be composed together into applications by end users.
- Granularity of JavaBeans components
  - Some JavaBean components will be used as building blocks in composing applications.
  - Some JavaBean components will be more like regular applications, which may then be composed together into compound documents

JavaBeans

- What is a Bean?
  - **"A Java Bean is a reusable software component that can be manipulated visually in a builder tool."**.
- Unifying features supported
  - "introspection" so a builder tool can analyze how a bean works.
  - "customization" so users can customize appearance and behaviour.
  - "events" as a simple communication metaphor than can be used to connect up beans.
  - "properties", both for customization and for programmatic use.
  - "persistence", so a bean can be customized and have its customized state saved away and reloaded later.

Goal

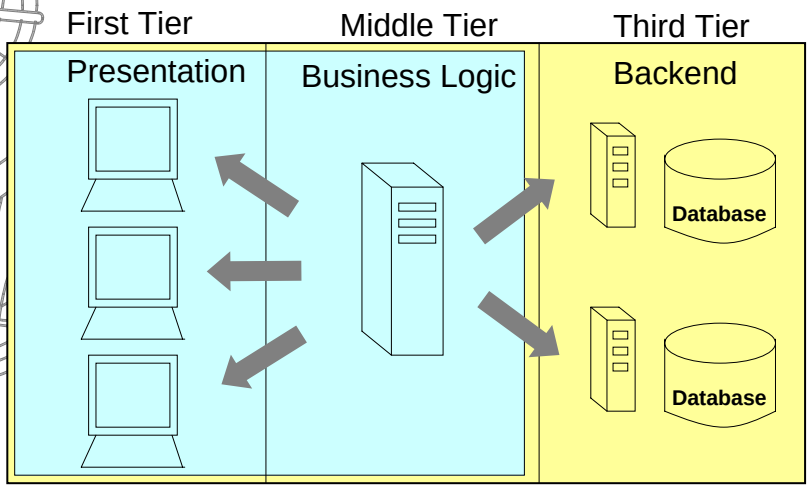
Let developer focus on the business logic

In the context of EJB the notion of component is played by ‘beans’ that implement certain business logic.

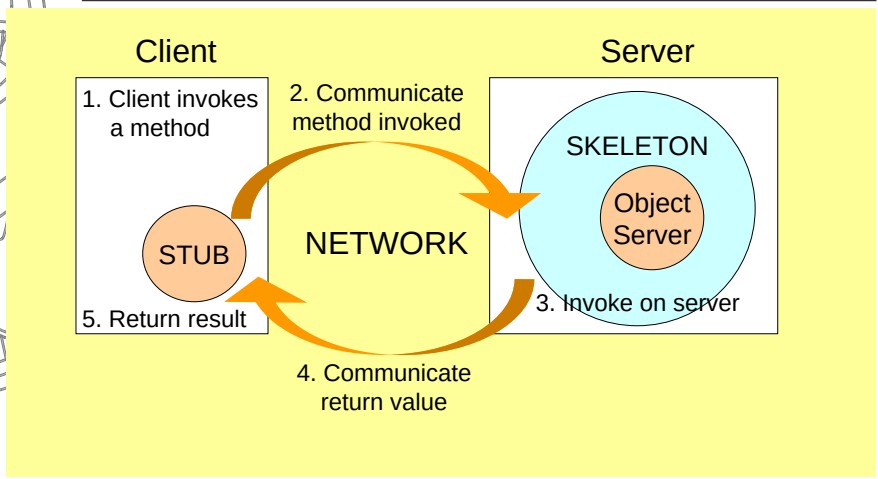
An EJB provides:

- mechanisms that allows beans to be deployed in a distributed setting
- managing services such as transactions, persistence, concurrency, and security

Context: Three-tier architectures



Implementing Distributed Objects using Stubs and Skeletons



Encapsulation of business logic

OO languages are used to improve development of GUIs, simplify access to data and to encapsulate business logic.

By encapsulating business models into objects you increase:

- flexibility
- extensibility
- reusability

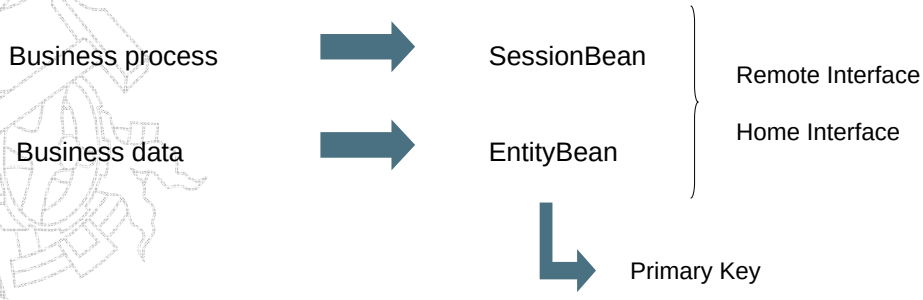
…and therefore the software can evolve as the business evolves

There are 2 types of business logic:

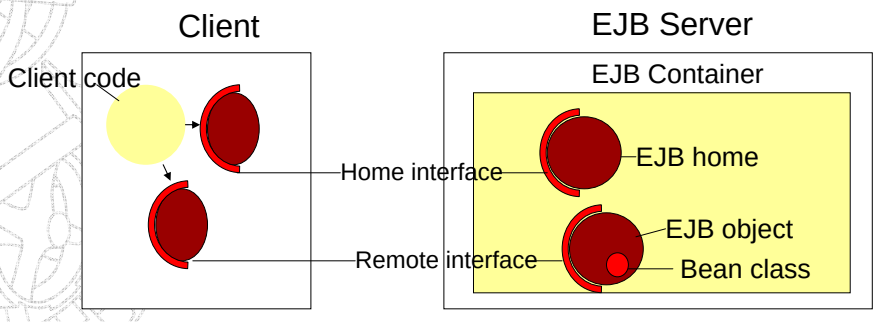
- Business processes
- Business data

Different types within business logic

A bean implements business logic:



Enterprise JavaBeans architecture



Red: define your self
Purple: automatically generated

Enterprise JavaBeans architecture

To implement an enterprise bean you need to define two interfaces and one or two classes :

- Interfaces:
  - Home interface
  - Remote interface
- Classes
  - Bean class
  - Primary key class (only for the entity bean)

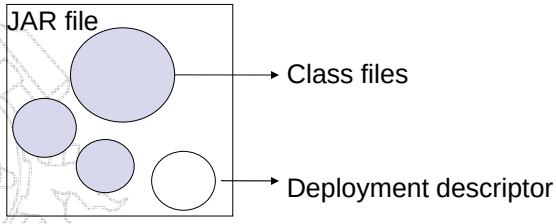
The client never interacts with a bean class directly; it always uses the methods of the bean's home and remote interfaces to do its work, interacting with the stubs that are generated automatically.

Enterprise JavaBeans architecture

- There are a lot of interactions between a bean and its server.
- These interactions are managed by a container, which is responsible for presenting a uniform interface between the bean and the server.
- The container is responsible for creating new instances of beans, making sure that they are stored properly by the server.
- Tools provided by the container's vendor do a tremendous amount of work behind the scenes.
- At least one tool will take care of creating the mapping between entity beans and records in your database.
- Other tools generate a lot of code based on the home interface, the remote interface, and the bean class itself.
- The code generated does things like create the bean, store it in the database, and so on.
- This code is what actually implements the Home and Remote interface, and is the reason your bean class only has to implement the business methods.

Bean deployment

Beans are deployed using JAR (Java ARchive) files.



A JAR file containing one or more enterprise beans include the bean classes, remote interfaces, home interfaces, and primary keys (Entity Beans only), for each bean. It also contains one deployment descriptor, which is used for all the beans in the JAR file. When a bean is deployed, the JAR's path is given to the container's deployment tools, which read the JAR file. The container uses the deployment descriptor to learn about the beans contained in the JAR file.

Resource Management

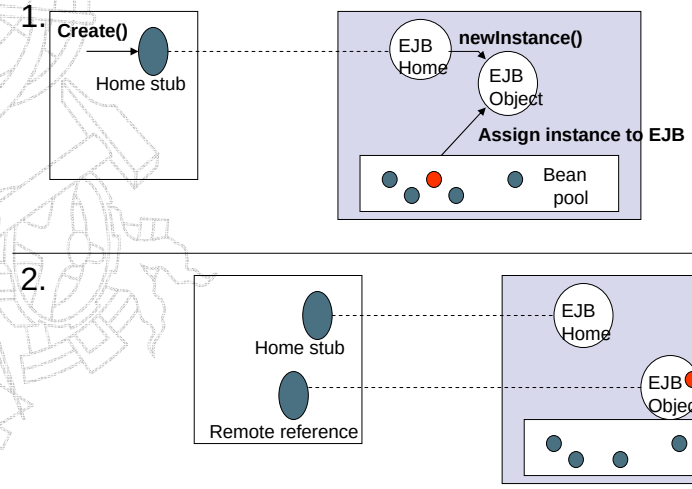
One of the benefits of EJB servers is that they are able to handle heavy workloads while maintaining a high level of performance.

EJB servers increase performance by

- synchronizing object interactions
- sharing resources

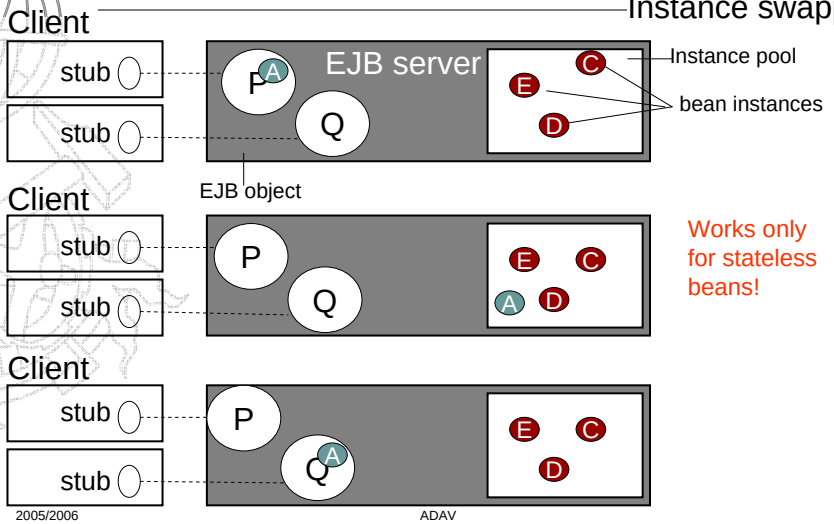
Resource Management

Instance pooling



Resource Management

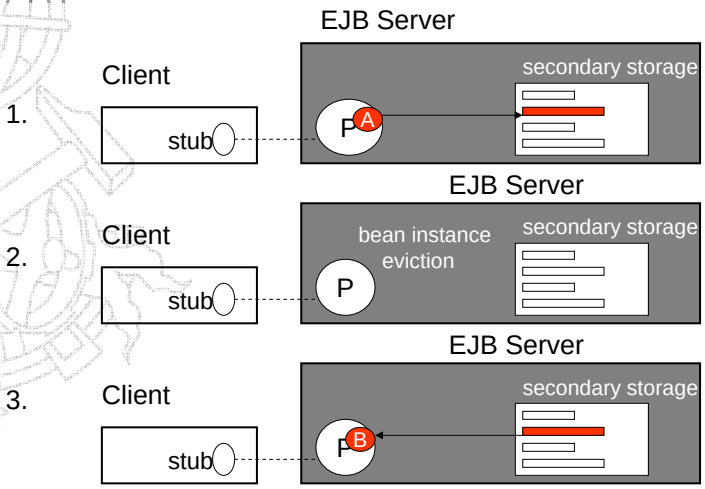
Instance swapping



Works only for stateless beans!

Resource Management

Bean Instance activation and passivation

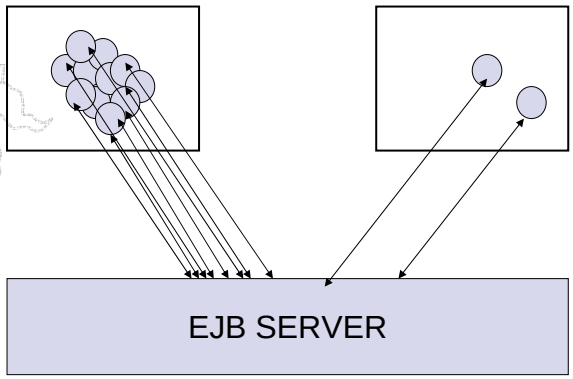


Resource Management

Session beans reduce network traffic and thin down clients

Client using only Entity beans
- Logic is in client
- Clients does lots of small updates on entity beans

Client using Session beans
- Logic is in the session beans
- Client only gives small number of commands to Session beans



Conclusion EJB is about hiding

EJB hides a lot of things for the developer. Such that the developer can focus on the business logic of the application.

EJB hides:

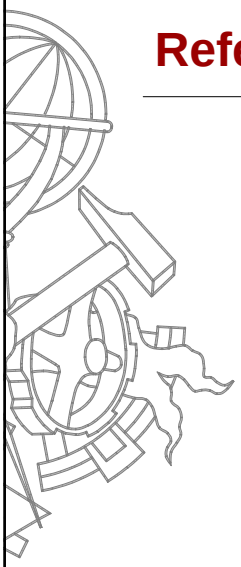
- Distribution of objects
- Resource Management
- Transactions
- Security
- Concurrency

So the developer doesn't have to know about skeletons, stubs, creation of beans, etc. He will only define the Home interface (for creation) and the Remote Interface (implemented) by the bean and can then focus on implementing the business logic.

Enterprise JavaBeans properties

Control Flow:	Managed by EJB container / EJB server
	Beans are passive (do a transaction and stop)
Distribution:	Bean resides on a Enterprise Bean Server
Topology:	Dynamic (bean creation/destruction/replacement/activation/…)
Interaction Style:	Remote Method Invocation
Binding Time:	Binding at Run Time
Binding Type:	External and Internal binding (Client and bean can do binding)
Multiplicity:	Multiple occurrences of a bean can be created.

References



Enterprise Java Beans (2nd ed) (2000)

Richard Monson-Haefel

ISBN: 1-56592-869-5

COM – Enterprise Java Beans - .NET

- Deploy the class as a Dynamic Link Library or Executable
- Deploy bean as a JAR file
- Deploy assembly as a Dynamic Link Library or Executable
- Notion of interfaces and implementations
- Notion of interfaces (Remote Interface & Home Interface) and implementation (bean)
- Notion of exposed methods (Described in Web Service Description Language)
- Use abstract base class to define binary interfaces
- Enterprise Java Bean Server / Container specification
- Use intermediate language, common type system, common language specification

COM – Enterprise Java Beans - .NET

- Techniques for dynamically selecting implementations
- Use Java Naming and Directory Interface for locating Beans (actually home object)
- DISCO (Discovery of Web Services) protocol
- Dynamic discovery of implemented interfaces
- ??? Not applicable because you never talk to bean directly.
- ??? Use Web Service Description Language

Agenda

Comparison of component models (COM, EJB and .NET)

Which component model serves you best?

Depending on your wishes / requirements you can choose the component model that serves you best. In this presentation I will discuss some criteria which can influence your choice.

For example:

- Performance
- Language independence
- Platform independence
- Desire to use existing components
- Type of application

Performance

Java byte code needs interpretation.
Network/Server overhead is large.
(when # beans increases there are optimalizations)

Binary code.
Little overhead on method call.
(especially on local components)

COM	++
EJB	-
.NET	+-

IL code needs compilation.
Performance of web services will depend on the used protocols and formats.

Language independence

C++, JAVA, VB, etc can be mixed

COM	+
EJB	-
.NET	+

Just Java

C++, C#, VB, etc can be mixed

OS independence

Due to the fact that Java is OS independent EJB is also OS independent.

Works on different platforms Components of one platform cannot be used on another.

COM	+-
EJB	++
.NET	+-

I see no fundamental reason why .NET could not be OS independent, however currently it is very Microsoft Windows oriented.

Reusing existing components

All the component models support easy reuse of components, this is basically what they were designed for !

.NET contains a large class library containing standard component you can reuse.

COM	+
EJB	+
.NET	++

Type of application

Embedded systems → COM

Transaction systems → EJB

Web application → .NET

Database application → EJB / .NET

Graphical application → .NET

Conclusions 1/3

Component models try to give solutions for the following problems:

- Reusability
  - Language / Platform independence
  - Separation of interface and implementation
- Distribution
  - Deployment of components
  - Location of components
  - Take care of communication
- Easy and Fast development
  - Providing standard components / functionality

Conclusions 2/3

- Component models have differences:
  - Language / Platform independence
    - Binary interface, Intermediate language
    - IDL, Common Language Specification, EJB container specification
  - Separation of interface and implementation
    - Explicit interfaces (COM, EJB) and more implicit interfaces (.NET)
  - Exploration of component features
    - QueryInterface, Deployment descriptor, Webservice description
  - Deployment of components
    - Exe, Dll, JAR
  - Location of components
    - Registry, Java Naming and Directory Interface, Disco files

Conclusions 3/3

Component models have differences (part II):

- Communication
  - MS RPC, Java RMI and default protocols like HTTP, SMTP in
  - combination with SOAP.
- Standard components / functionality
  - Communication, database, security, .NET Framework class library

The choice of the component model you will use will depend on the type of application you are developing and the requirements on the application.

Questões

?