Enterprise Java Beans
Taking IT easy with Middleware

Premkumar Devanbu,
Center for Software Systems Research
Dept. of Computer Science,
University of California, Davis, CA, USA.
Outline

• Introduction, Motivation and Overview.
• Session Beans
• Entity Beans
• EJB Services: Naming, Transactions, and Security.
• Comparison of EJB and .NET
Outline

• Introduction, Motivation and Overview.
• Session Beans
• Entity Beans
• EJB Services: Naming, Transactions, and Security.
• Comparison of EJB and .NET
Distributed IT Systems are ubiquitous

- Web e-business systems
- Web Intranet systems (e.g., personnel, registration, grading, etc at a University.)
- Government systems (e.g., taxation, welfare programs, research grants)
......and challenging!!

- Networking
- Administration
- Fault-tolerance
- Persistence
- Transactions
- Load handling

- Security
- Resource pooling/Caching Replacement
- Interoperability
- Legacy systems
- Logging/auditing
They do have similarities!!

Application-level commonalities:

- Login/authentication
- Pricing computations (can be complex!)
- Session logic (e.g., shopping cart)
- Tax Computations
- Payment/Credit card
They share infra-structure!

Typical common elements:
- Data-base management
- Protocol handlers (TCP/IP)
- Security (cryptography, key management).
- Transaction management.
- Load management/scheduling

...So component-based approach is very appealing!

...But! How to provide an environment for a component market to thrive?
What we would like:

• A common architectural style, with good separation of concerns.
• A operational infra-structure (i.e., a container) that provides good services to components
• Well-defined, standardized, interfaces for components
• Tool support (automate the repetitive tasks)

...provided by EJB, .NET, CCM etc.
We’ll start with:

- A common architectural style, with good separation of concerns.
- A operational infra-structure (i.e., a container) that provides good services.
- Well-defined, standardized, interfaces
- Tool support (automate the repetitive tasks)
The Architectural Style

• Should be simple, yet general.
• Should provide good separation of concerns.
  - To support easier evolution
  - To support many different vendors/systems
• Simplify personnel/training issues.
Evolution of Container Style

...In the beginning: (Simple Client-Server)

- Proprietary Protocols
  - Complex Clients
  - Complex Servers
  - Commitment to vendor
  - Hard to evolve.
  - Hard to administer
Evolution of Container Style

...Things get better (RDB/xDBC)

- Data management separated.
- Easier to scale database server
- Application logic still complex
- Hard to evolve applications
- Still have Complex Clients
- Hard to administer clients
Evolution of Container Style

...here come Servlets

- Clients much simpler
- Trivial to administer
- Servlet becomes very complex:
  - GUI
  - Persistence
  - Business Logic, etc.
Introducing The EJB Style....

- Business Logic handled by EJBs.
- Database persistence also by EJBs.
- GUI concern handled by servlets.
What are these containers?

Handle common functionality for all the contained things.

- Communications.
- Threading, stopping/starting
- Resource pooling
- Global/session state
What do containers do?

**Servlet Container:**
- HTTP Protocol
- Web Request/response
- Threading
- Servlet/session state
- Cookies

**EJB Container**
- Communications.
- Threading/Start/Stop
- Resource pooling
- Transactions
- Security
- Data Persistence, etc.

© P. Devanbu, 2002
Further refining….

**Servlet Container**

**EJB Container**

**JDBC**

HTTP

**Session Beans**

Handle each user’s session, and contain the business logic.

**Entity Beans**

Handle the storage/retrieval of data, and any other data access and querying functionality.

**J2EE framework** includes both containers!

© P. Devanbu, 2002
What is an Enterprise Java Bean?

A server-side component that provides some unit of functionality, and is compliant with interfaces that allow it to operate with the J2EE container. Implemented as a collection of Java classes, that implement certain required interfaces.
Examples

A typical purchase transaction:
- A shopping cart EJB
- A credit card EJB (remote, at Visa/Amex)
- A shipping EJB
- An inventory control EJB
The EJB Universe

Tools Provider

Bean Provider

EJB Platform Vendors

EJB Platform Implementation

Application Package

System Operator

Tools

EJBs

Application

Deployment Specialist

Application Integrator

© P. Devanbu, 2002
Moving ahead:

• A common architectural style, with good separation of concerns.
• A operational infra-structure (i.e., a container) that provides good services.
• Well-defined, standardized, interfaces
• Tool support (automate the repetitive tasks)

...provided by EJB, .NET, CCM etc.
A typical “old-style” component business method

Transfer(amount, account1, account2) {
    /* 1. Do security...authenticate account */
    /* 2. Signal beginning of transaction */
    /* 3. Access Database for account1 */
    /* 4. Access Database for account2 */
    /* 5. Update accounts 1 and 2 in memory */
    /* 6. Write out account1 to database */
    /* 7. Write out account2 to database */
    /* 8. Signal Commit of Transaction */
}

Programmer must handle persistence, transactions, security...
With Container-managed security, persistence and Transactions...

Transfer(amount, account1, account2) {

/* 1. Do security...authenticate account */
/* 2. Signal beginning of transaction */
/* 3. Access Database for account1 */
/* 4. Access Database for account2 */
/* 5. Update accounts 1 and 2 in memory */
/* 6. Write out account1 to database */
/* 7. Write out account2 to database */
/* 8. Signal Commit of Transaction */
}

All other Details automatically taken care of by container!!!
Is this Magic? No!

• Container performs the services, under instructions from deployment specialist!
• Container performs services by “intercepting” bean operations. OBSERVER pattern is used, along with DECORATOR.
• Programmer can customize some things by programming if so desired, and tools generate code in other cases. TEMPLATE pattern.
How The Container intervenes
The process

• Write EJB, with just business logic.
• Describe services required, in a separate deployment descriptor.
• Run a Tool, which generates “interceptor” code that is run by the remote proxy.
• The “interceptor” code handles all the calls to the middleware-provided services.
Advantages

• Application/business logic is much simpler.
• Separation of concerns.
• Easy to change use of services (transactions, security, persistence) without re-programming.