Patterns for EJB Development

Bobby Woolf
Independent Consultant
Cyberdyne Software, Inc.
woolf@acm.org

Copyright 2002, Cyberdyne Software, Inc.

Contents

- Layered Application Architecture
- The Patterns
- Cookbook
- Bonus Patterns
- Related Trends in EJB

Copyright 2002, Cyberdyne Software, Inc.

Contents: Layered Application Architecture

- **Layered Application Architecture**
- The Patterns
- Cookbook
- Bonus Patterns
- Related Trends in EJB

Copyright 2002, Cyberdyne Software, Inc.

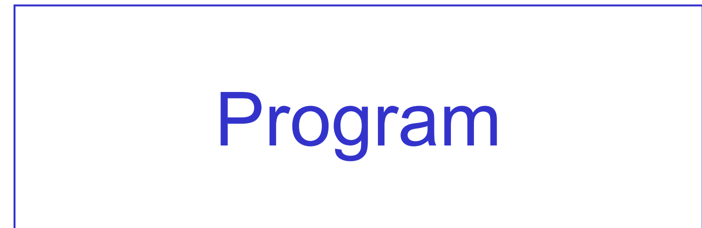
Layered Application Architecture

- Early Layered Architectures
- Four Layered Architecture
- Service Layer Architecture
- Distributed Architecture
- Tiered Architecture
- J2EE Architecture

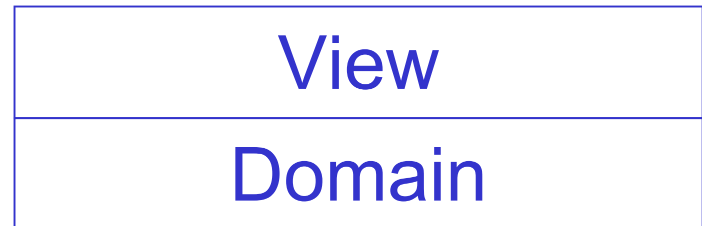
Copyright 2002, Cyberdyne Software, Inc.

Early Layered Architectures

- First architecture: No layers
  - One big block of code



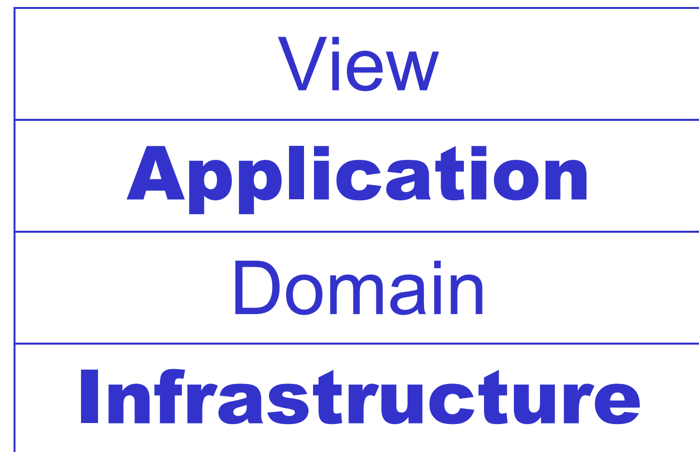
- Second architecture: Two layers
  - Separate presentation from domain



Copyright 2002, Cyberdyne Software, Inc.

Four Layer Architecture

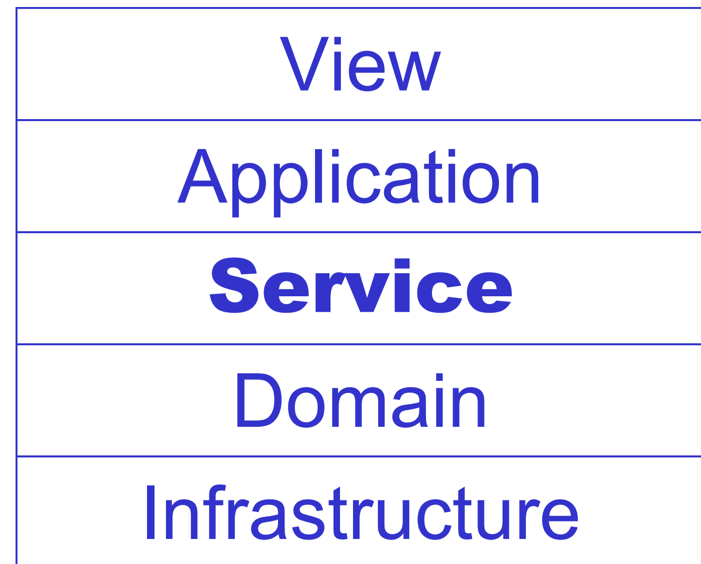
- Next architecture: Four layers
  - View/Presentation
    - Swing, HTML, etc.
  - Application (Controller)
    - View’s Model
  - Domain (Model)
    - Business Model
  - Infrastructure/Persistence
    - Database, legacy, etc.



Copyright 2002, Cyberdyne Software, Inc.

Service Layer Architecture

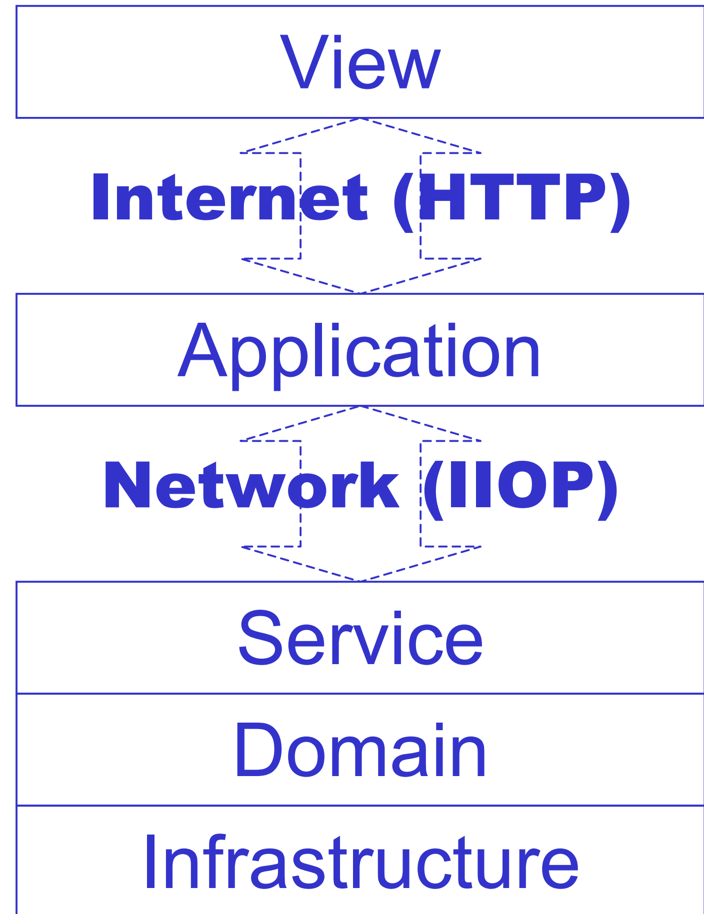
- Latest architecture: Five layers
- Service Layer
  - Menu of services
  - Encapsulates use cases
  - Controls transactions
  - Channels network chatter
  - Coordinates domain objects



Copyright 2002, Cyberdyne Software, Inc.

Distributed Architecture

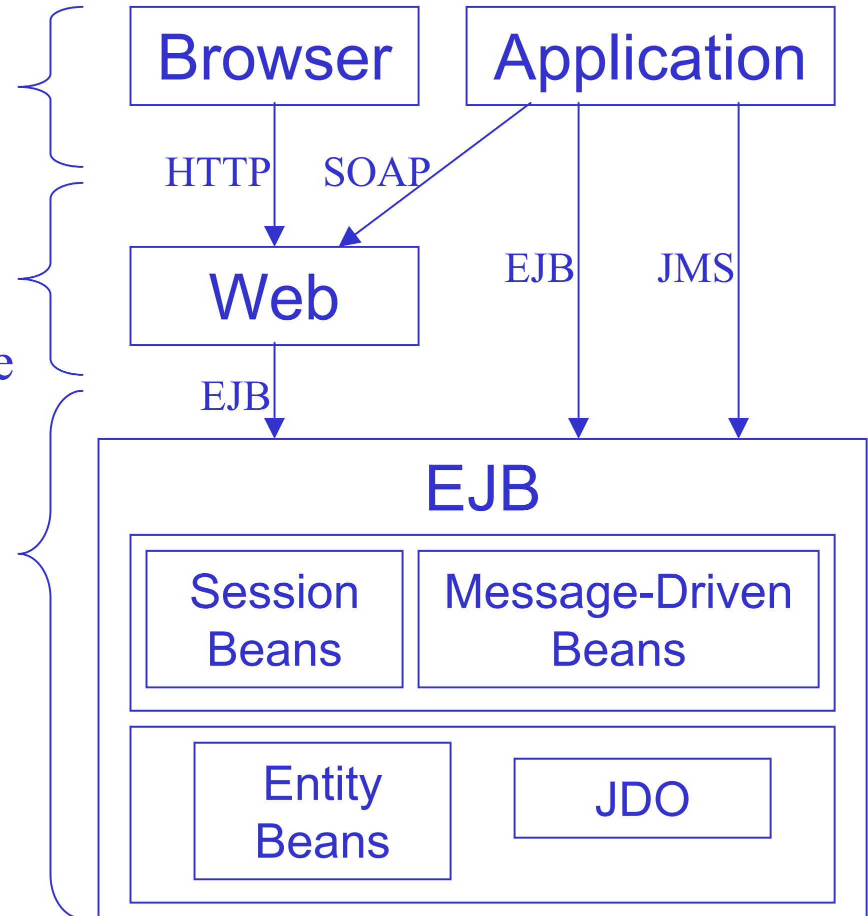
- Layers enable distributed architectures
  - View
    - HTML, JavaScript, web
  - Application
    - Servlets/JSP
  - Service
    - Session, message beans
  - Domain, Infrastructure
    - Entity beans



Copyright 2002, Cyberdyne Software, Inc.

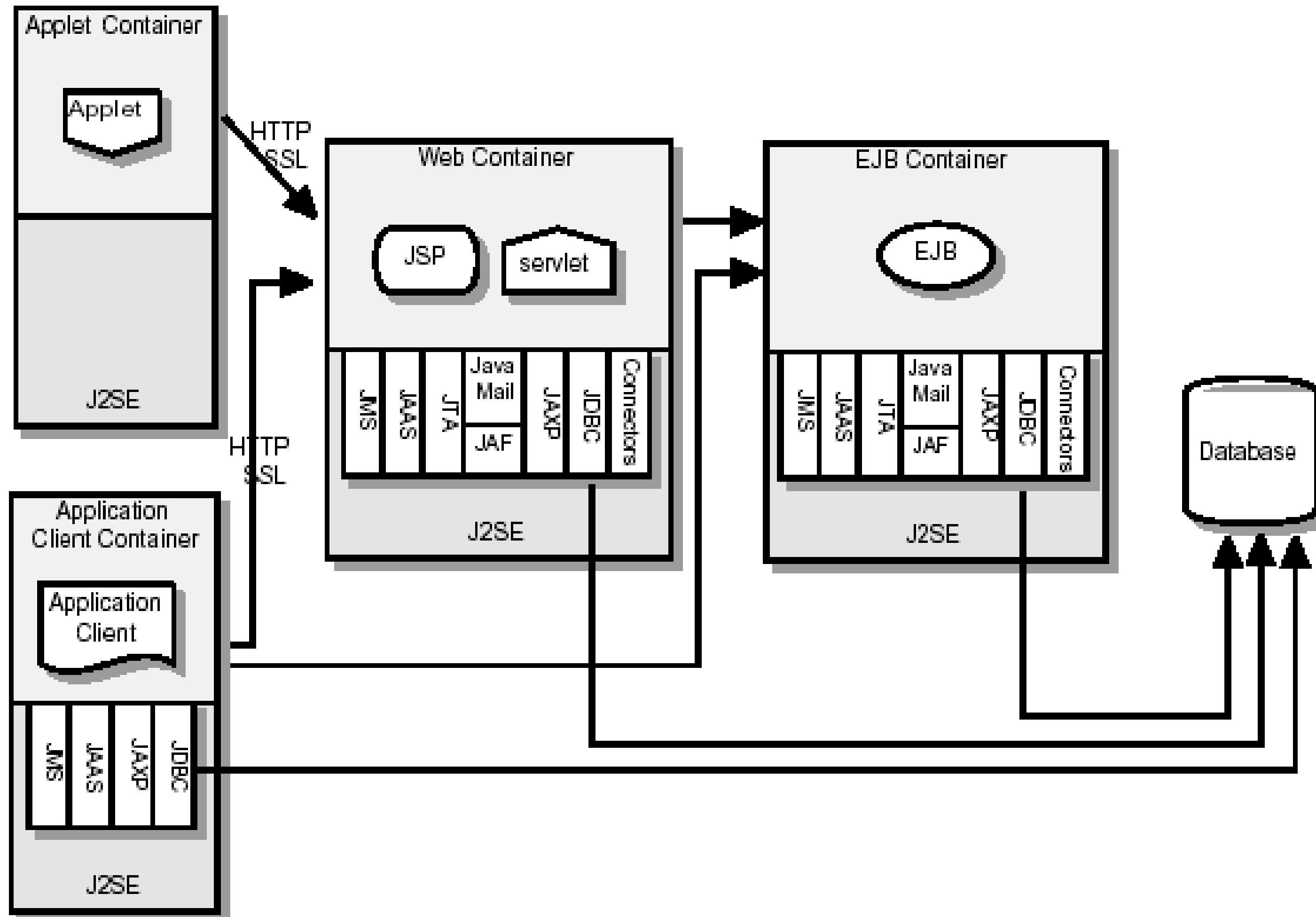
Tiered Architecture

- Layers deploy in tiers
  - Client tier
    - Web browser or J2SE
  - Web tier
    - Servlet/JSP/SOAP engine
  - Business tier
    - EJB/J2EE container (application server)
  - EIS tier *(not shown)*
    - Enterprise information systems



Copyright 2002, Cyberdyne Software, Inc.

J2EE Architecture



Layered Application Architecture: Review

- Early Layered Architectures
- Four Layered Architecture
- Service Layer Architecture
- Distributed Architecture
- Tiered Architecture
- J2EE Architecture

Copyright 2002, Cyberdyne Software, Inc.

Contents: The Patterns

- Layered Application Architecture
- **The Patterns**
- Cookbook
- Bonus Patterns
- Related Trends in EJB

Copyright 2002, Cyberdyne Software, Inc.

The Patterns

- Service Bean
- Services Package
- Bean Service Interface
- Remotified Interface
- Unremotified Bean Class
- Service Failed Exception
- Application Package
- Application Service Interface

Copyright 2002, Cyberdyne Software, Inc.

Basic EJB Patterns

- Remote Interface
  - What behavior does the bean provide a client?
  - Example: **Item**
- Home Interface
  - How do clients access bean instances?
  - Example: **ItemHome**
- Bean Class
  - How does a bean implement its interface?
  - Example: **ItemBean**

Copyright 2002, Cyberdyne Software, Inc.

The Patterns: Service Bean

- **Service Bean**
- Services Package
- Bean Service Interface
- Remotified Interface
- Unremotified Bean Class
- Service Failed Exception
- Application Package
- Application Service Interface

Copyright 2002, Cyberdyne Software, Inc.

Service Bean: Problem

- What sort of interface should an EJB present to its clients?
- Forces:
  - Client needs an easy way to request work
  - Work should be bundled into units
    - One transaction
    - One network call
    - One security check
  - Units of work should be easily reusable

Copyright 2002, Cyberdyne Software, Inc.

Service Bean: Solution

- Structure the EJB as a *Service Bean*
  - A set of services the bean can perform
  - Each service a use case
  - Related use cases provided by a single bean
  - Avoid getters and setters

Copyright 2002, Cyberdyne Software, Inc.

Service Bean: Consequences

- Client makes a single call across the network
- A single security check is made
- Unit of work is performed in a single transaction
- Implementation is hidden from the client
  - Implementation can easily be changed

Copyright 2002, Cyberdyne Software, Inc.

Service Bean: Examples

- AddressBookService
  - public void createPerson(fName, lName, …)
  - public Person find(fName, lName)
  - public void update(fName, lName, …)
  - public void delete(fName, lName)
- BankService
  - public void transferMoney(amt, acct1, acct2)
  - public Money getBalance(acct)

Copyright 2002, Cyberdyne Software, Inc.

Service Bean: AKA

- Also known as: Service Layer
  - A common set of application operations available to multiple kinds of clients
  - Coordinates the application's response in each operation
- Also known as: Session Façade
  - Separates presentation and business logic
  - Business-tier controller

Copyright 2002, Cyberdyne Software, Inc.

The Patterns: Services Package

- Service Bean
- **Services Package**
- Bean Service Interface
- Remotified Interface
- Unremotified Bean Class
- Service Failed Exception
- Application Package
- Application Service Interface

Copyright 2002, Cyberdyne Software, Inc.

Services Package: Problem

- What package should Service Beans go in?
- Forces:
  - Service beans promote client/server code
  - Put separate layers of code in separate packages
  - Separate client and server packages will help create separate client and server jar files

Services Package: Solution

- Create a *Services Package*
- Make it a subpackage named "services"
  - Example: **addressbook.services**
- Put the service beans in the service package
  - **AddressBook**, **AddressBookService**, **AddressBookHome**, **AddressBookBean**
- Next patterns:
  - Bean Service Interface, Remote Interface, Home Interface, and Bean Class

Copyright 2002, Cyberdyne Software, Inc.

Services Package: Example

- com.myco.myproj.addressbook.services
  - AddressBook
  - AddressBookService
  - AddressBookHome
  - AddressBookBean
- Javadoc: Services package shows the services available to the client
  - Just browse addressbook.services

Copyright 2002, Cyberdyne Software, Inc.

The Patterns: Bean Service Interface

- Service Bean
- Services Package
- **Bean Service Interface**
- Remotified Interface
- Unremotified Bean Class
- Service Failed Exception
- Application Package
- Application Service Interface

Copyright 2002, Cyberdyne Software, Inc.

Bean Service Interface: Problem

- How can the compiler ensure that the Remote Interface and Bean Class are consistent?
- Forces:
  - Class must implement interface's methods
  - Class can **not** implement **EJBObject** methods
  - Class does **not** implement the Remote Interface
  - Compiler needs relationship between class and interface

Copyright 2002, Cyberdyne Software, Inc.

Bean Service Interface: Solution

- Declare the remote bean messages in a *Bean Service Interface*
  - Remote Interface will extend this and EJBObject
  - Bean Class will implement this interface
    - As well as SessionBean or EntityBean
  - Compiler will ensure consistency

Copyright 2002, Cyberdyne Software, Inc.

Bean Service Interface: Example

- AddressBookService
  - Declares service methods
- AddressBook
  - Remote Interface
  - Extends AddressBookService, EJBObject
- AddressBookBean
  - Bean Class
  - Implements AddressBookService, SessionBean

Copyright 2002, Cyberdyne Software, Inc.

Bean Service Interface: AKA

- Also known as:
  - Implementing a Common Interface
  - Business Interface
- Next Patterns:
  - Bean Service Interface ia a Remotified Interface
  - Bean Service Interface messages throw a Service Failed Exception

Copyright 2002, Cyberdyne Software, Inc.

The Patterns: Remotified Interface

- Service Bean
- Services Package
- Bean Service Interface
- **Remotified Interface**
- Unremotified Bean Class
- Service Failed Exception
- Application Package
- Application Service Interface

Copyright 2002, Cyberdyne Software, Inc.

Remotified Interface: Problem

- How to design an EJB interface to be used remotely?
- Forces:
  - Bean Service, Remote, and Home Interfaces are invoked remotely
  - Each method must throw RemoteException
  - A consequence of java.rmi.Remote
  - Container throws RemoteException

Copyright 2002, Cyberdyne Software, Inc.

Remotified Interface: Solution

- Implement interface as a *Remotified Interface*
  - Each method throws, at least, RemoteException
- Implement a business interface as usual
  - Add RemoteException to each method signature
- No need to extend Remote
  - The Remote Interface already does

Copyright 2002, Cyberdyne Software, Inc.

Remotified Interface: Consequences

- Remotified Interface is not quite like a business interface
  - Must throw RemoteException, a checked exception
  - Clients must catch and handle Remote-Exception, even when client is not remote
- Only remotify when required
  - RMI, CORBA, and EJB require it

Remotified Interface: Example

- AddressBookService
  - createPerson(…) **throws RemoteException**
  - find(…) **throws RemoteException**
  - update(…) **throws RemoteException**
  - delete(…) **throws RemoteException**

Copyright 2002, Cyberdyne Software, Inc.

The Patterns: Unremotified Bean Class

- Service Bean
- Services Package
- Bean Service Interface
- Remotified Interface
- **Unremotified Bean Class**
- Service Failed Exception
- Application Package
- Application Service Interface

Copyright 2002, Cyberdyne Software, Inc.

Unremotified Bean Class: Problem

- Should bean class methods be remotified?
- Forces:
  - Class can throw fewer exceptions than interface
  - Only exceptions this implementation can throw
  - Bean Class should not throw RemoteException
    - Only containers throw RemoteException
  - Don't use RemoteException for server logic problems
    - Only for communications problems

Copyright 2002, Cyberdyne Software, Inc.

Unremotified Bean Class: Solution

- Implement bean class as an *Unremotified Bean Class*
  - Methods do **not** throw RemoteInterface
    - Method signatures don't declare it
    - Method implementations don't do it
- Consequence:
  - Throw a Service Failed Exception instead

Unremotified Bean Class: Example

- AddressBookBean
  - createPerson(…) **throws** ***ServiceFailedException***, ~~**RemoteException**~~
  - find(…) **throws** ***ServiceFailedException***, ~~**RemoteException**~~
  - update(…) **throws** ***ServiceFailedException***, ~~**RemoteException**~~
  - delete(…) **throws** ***ServiceFailedException***, ~~**RemoteException**~~

Copyright 2002, Cyberdyne Software, Inc.

The Patterns: Service Failed Exception

- Service Bean
- Services Package
- Bean Service Interface
- Remotified Interface
- Unremotified Bean Class
- **Service Failed Exception**
- Application Package
- Application Service Interface

Copyright 2002, Cyberdyne Software, Inc.

Service Failed Exception: Problem

- What if a Service Bean cannot perform a service successfully?
- Forces:
  - Methods indicate failure with exceptions
  - Indicate context problems, not coding problems
  - Multiple, unrelated exceptions problematic
    - Many for client to handle
    - Expose implementation
  - Just indicate that service failed

Copyright 2002, Cyberdyne Software, Inc.

Service Failed Exception: Solution

- Method should throw one type of exception
  - A *Service Failed Exception*
  - Indicates that service failed and which service failed
  - Exception can have subclasses for specifics
- Each Bean Class method should only throw one type of exception
  - Exceptions should be caught, wrapped, and rethrown

Copyright 2002, Cyberdyne Software, Inc.

Service Failed Exception: Single Exception Example

- Don’t do this:
  - AddressBookBean
    - createPerson(…) throws **IOException**, **SQLException**, **SAXException**, **ClassNotFoundException**, **CloneNotSupportedException**
- Instead, do this:
  - AddressBookBean
    - createPerson(…) throws **CreatePersonException**

Copyright 2002, Cyberdyne Software, Inc.

Service Failed Exception: Service Bean Example

- AddressBookService
  - createPerson(…) **throws CreatePersonException**, **RemoteException**
  - find(…) **throws FindPersonException**, **RemoteException**
  - update(…) **throws UpdatePersonException**, **RemoteException**
  - delete(…) **throws DeletePersonException**, **RemoteException**

Copyright 2002, Cyberdyne Software, Inc.

Service Failed Exception: Bean Class Example

- AddressBookBean
  - createPerson(…) **throws CreatePersonException**
  - find(…) **throws FindPersonException**
  - update(…) **throws UpdatePersonException**
  - delete(…) **throws DeletePersonException**

Copyright 2002, Cyberdyne Software, Inc.

Service Failed Exception: Wrapped Exception Design

- AddressBookException extends Exception
  - Create-, Find-, Update-, DeletePersonException extend AddressBookException
- AddressBookException wrappers other exceptions
  - Instance variable of type Exception
  - Constructor with Exception parameter
- Reusable WrappedException class

Copyright 2002, Cyberdyne Software, Inc.

Service Failed Exception: Wrapped Exception Example

- AddressBookBean

```
createPerson(...) throws CreatePersonException {
   try {
         ...
   }
   catch (Exception ex) {
         throw new CreatePersonException(ex);
   }
}
```

Copyright 2002, Cyberdyne Software, Inc.

Patterns in EJB Home

- EJB Home
  - Service Bean for managing beans
  - EJB Home Interface is a Remotified Interface
    - Each method throws RemoteException
  - EJB Home Interface throws Service Failed Exceptions
    - create throws CreateException
    - find throws FinderException
    - remove throws RemoveException

Copyright 2002, Cyberdyne Software, Inc.

The Patterns: Application Package

- Service Bean
- Services Package
- Bean Service Interface
- Remotified Interface
- Unremotified Bean Class
- Service Failed Exception
- **Application Package**
- Application Service Interface

Copyright 2002, Cyberdyne Software, Inc.

Application Package: Problem

- What package do Service Bean clients go in?
- Forces:
  - Same package as the beans they use?
  - Put separate layers of code in separate packages
  - Client code should be separate from server code
  - Separate client and server packages will help create separate client and server jar files

Copyright 2002, Cyberdyne Software, Inc.

Application Package: Solution

- Create an *Application Package*
- Make it a subpackage named "application"
  - Example: **addressbook.application**
- Put the service beans in the service package
  - **AddressBook**, **AddressBookEjbClient**
- Next patterns:
  - Application Service Interface
    - One per Service Bean

Copyright 2002, Cyberdyne Software, Inc.

Application Package: Example

- com.myco.myproj.addressbook.application
  - AddressBook
  - AddressBookEjbClient
- Javadoc: Application package shows the services available on the client
  - Just browse addressbook.application
  - Client-side services that are already implemented

Copyright 2002, Cyberdyne Software, Inc.

The Patterns: Application Service Interface

- Service Bean
- Services Package
- Bean Service Interface
- Remotified Interface
- Unremotified Bean Class
- Service Failed Exception
- Application Package
- **Application Service Interface**

Copyright 2002, Cyberdyne Software, Inc.

Application Service Interface: Problem

- How should clients access a service bean?
- Forces:
  - Simple way to access a Service Bean’s services
  - An EJB client has to know about EJB
    - Use Remote Interface
    - Handle RemoteExceptions
  - Encapsulate/hide details of EJB, remoteness

Copyright 2002, Cyberdyne Software, Inc.

Application Service Interface: Solution

- Represent the service on the client as an Application Service Interface
- Implementors encapsulate EJB details
  - Hide whether EJB is even being used
  - Hide home lookup, bean access
  - Hide remoteness issues, RemoteException
  - Allow for alternative implementations
    - Stand-alone, EJB, SOAP/web services, etc.

Copyright 2002, Cyberdyne Software, Inc.

Application Service Interface: Consequences

- Application Service Interface is very much like Bean Service Interface
  - Same methods
  - App interface doesn’t throw RemoteException
  - Does throw Service Failed Exceptions
- Application Service Interface is unremotified
  - Like Unremotified Bean Class

Copyright 2002, Cyberdyne Software, Inc.

Application Service Interface : AKA

- Also known as: Business Delegate
  - Hides implementation details of business service
  - Expose simpler, uniform interface
- Delegation
  - An object forwards a request to another object
- Also known as:
  - Access Beans (IBM WebSphere)

Copyright 2002, Cyberdyne Software, Inc.

Service Interfaces Comparison

- Bean Service Interface
  - addressboook.**services**.AddressBookService
    - createPerson(…) throws CreatePersonException, **RemoteException**
    - find(…) throws FindPersonException, **RemoteException**
- Application Service Interface
  - addressbook.**application**.AddressBook
    - createPerson(…) throws CreatePersonException
    - find(…) throws FindPersonException

Copyright 2002, Cyberdyne Software, Inc.

The Patterns: Review

- Service Bean
- Services Package
- Bean Service Interface
- Remotified Interface
- Unremotified Bean Class
- Service Failed Exception
- Application Package
- Application Service Interface

Copyright 2002, Cyberdyne Software, Inc.

Contents: Cookbook

- Layered Application Architecture
- The Patterns
- **Cookbook**
- Bonus Patterns
- Related Trends in EJB

Copyright 2002, Cyberdyne Software, Inc.

Cookbook

- Create Exceptions
- Define Application Service Interface
- Define Bean Service Interface
- Define Remote Interface
- Implement Bean Class
- Implement Application Service Interface

Copyright 2002, Cyberdyne Software, Inc.

Cookbook: Exceptions

- Create Exceptions first
  - Service Failed Exceptions

```
public class AddressBookException
        extends WrappedException {
        // constructors
  }
public class CreatePersonException
        extends AddressBookException {
        // constructors
  }
```

Copyright 2002, Cyberdyne Software, Inc.

Cookbook: Application Service Interface

- Goes in Application Package
- Each method: Service Failed Exception
  - No RemoteException

```
package addressbook.application;
import addressbook.services.CreatePersonException;
public interface AddressBook {
   public void createPerson
         (String firstName, String lastName)
         throws CreatePersonException;
}
```

Copyright 2002, Cyberdyne Software, Inc.

Cookbook: Bean Service Interface

- Goes in Services Package
  - Name: App. Service Interface + “Service”
  - Service Bean, Remotified Interface

```
package addressbook.services;
    import java.rmi.RemoteException;
    public interface AddressBookService {
            public void createPerson
                    (String firstName, String lastName)
                    throws CreatePersonException,
                    RemoteException;
    }
```

Copyright 2002, Cyberdyne Software, Inc.

Cookbook: Remote Interface

- Goes in Services Package
  - Name: Same as App. Service Interface
  - Implementation is already in Bean Service Interface

```
package addressbook.services;
    import javax.ejb.EJBObject;
    public interface AddressBook entends
            AddressBookService, EJBObject {
            // no additional messages
    }
```

- Also implement Home Interface

Cookbook: Bean Class

- Goes in Services Package
- Implements
  - Bean Service Interface
  - Appropriate extension of EnterpriseBean
- Unremotified Bean Class
  - No RemoteExceptions

Copyright 2002, Cyberdyne Software, Inc.

Cookbook: Bean Class

```
package addressbook.services;
   import javax.ejb.SessionBean;
   public class AddressBookBean implements
           AddressBookService, SessionBean {
           public void createPerson
                   (String firstName, String lastName)
                   throws CreatePersonException {
                   // code to create the Person in the
                   // Address Book
                   // any exceptions are wrapped as
                   // CreatePersonExceptions
           }
   }
```

Copyright 2002, Cyberdyne Software, Inc.

Cookbook: Application Service Implementor

- Implement the Application Service Interface
  - Implement as an EJB client
- Goes in Application Package

```
package addressbook.application;
   import addressbook.services.AddressBook;
   import addressbook.services.AddressBookHome;
   import addressbook.services.CreatePersonException
public class AddressBookEjbClient implements
            addressbook.application.AddressBook
```

Copyright 2002, Cyberdyne Software, Inc.

Cookbook: Application Service Implementor

```
public void createPerson
   (String firstName, String lastName)
   throws CreatePersonException {
        try {
          AddressBook book = getService();
          book.createPerson (firstName, lastName);
          book.remove();
        }
        catch (Exception ex) {
          throw new CreatePersonException(ex);
        }
   }
```

Copyright 2002, Cyberdyne Software, Inc.

Cookbook: Application Service Implementor

```
protected AddressBook getService()
   throws <Exceptions> {
        return getServiceHome.create();
   }
protected AddressBookHome getServiceHome()
   throws <Exceptions> {
        // code to obtain the home
   }
```

Cookbook: Review

- Create Exceptions
- Define Application Service Interface
- Define Bean Service Interface
- Define Remote Interface
- Implement Bean Class
- Implement Application Service Interface

Copyright 2002, Cyberdyne Software, Inc.

Contents: Bonus Patterns

- Layered Application Architecture
- The Patterns
- Cookbook
- **Bonus Patterns**
- Related Trends in EJB

Copyright 2002, Cyberdyne Software, Inc.

Bonus Patterns

- Data Transfer Object
- Identity Object
- Data Transfer Object Assembler

- Bonus Strategy
  - Stateful vs. Stateless Session Beans

Copyright 2002, Cyberdyne Software, Inc.

Bonus Patterns: Data Transfer Object

- **Data Transfer Object**
- Identity Object
- Data Transfer Object Assembler

- Bonus Strategy:
  - Stateful vs. Stateless Session Beans

Copyright 2002, Cyberdyne Software, Inc.

Data Transfer Object: Problem

- Also known as
  - Value Object
- How can client get bulk data from server without multiple network calls?
- Forces:
  - Multiple network calls hurt performance
  - Multiple parameters awkward; multiple return values impossible
  - Entity beans cannot be copied to the client

Copyright 2002, Cyberdyne Software, Inc.

Data Transfer Object: Solution

- Transfer bulk data between the client and server using a *Data Transfer Object*
  - Plain Java object
  - Dumb data object
  - Often aggregates data from multiple server objects
- Custom designed for a particular client
  - Don’t overload with unnecessary state
- Must be serializable

Copyright 2002, Cyberdyne Software, Inc.

Data Transfer Object: Example

- Person
  - instance variables
    - firstName, lastName, street, city, state, zip
  - getters and setters
- Scenario
  - Client requests Person
  - Server creates Person, sends to client
  - Client views, changes data, sends back to server
  - Server gets changes and commits them

Copyright 2002, Cyberdyne Software, Inc.

Bonus Patterns: Identity Object

- Data Transfer Object
- **Identity Object**
- Data Transfer Object Assembler

- Bonus Strategy:
  - Stateful vs. Stateless Session Beans

Copyright 2002, Cyberdyne Software, Inc.

Identity Object: Problem

- How can client choose from many domain objects?
- Forces:
  - Domain objects cannot be copied to client
  - Most choices will never be used
  - Server must know which choice was chosen
  - Minimize bandwidth, client memory consumption

Copyright 2002, Cyberdyne Software, Inc.

Identity Object: Solution

- Use an *Identity Object* to specify an object on the server
- Very lightweight
  - A name/identifier the user recognizes
  - A UID/primary key/foreign key the server recognizes
- Must be serializable
- Specialization of Data Transfer Object

Copyright 2002, Cyberdyne Software, Inc.

Identity Object: Example

- PersonIdentity
  - instance variables
    - name, personPrimaryKey
  - getters and setters
- Scenario
  - Client requests list of Persons
  - Server creates PersonIdentities, sends to client
  - Client views list, sends selection back to server
  - Server gets selection, returns DTO

Copyright 2002, Cyberdyne Software, Inc.

Bonus Patterns: Data Transfer Object Assembler

- Data Transfer Object
- Identity Object
- **Data Transfer Object Assembler**

- Bonus Strategy:
  - Stateful vs. Stateless Session Beans

Copyright 2002, Cyberdyne Software, Inc.

Data Transfer Object Assembler

- Problem
  - How to create Data Transfer Objects, convert Identity Objects into DTOs, etc.?
- Solution
  - Use a Data Transfer Object Assembler
- Also known as
  - Value Object Assembler

Copyright 2002, Cyberdyne Software, Inc.

Bonus Patterns: Stateful vs. Stateless

- Data Transfer Object
- Identity Object
- Data Transfer Object Assembler

- Bonus Strategy:
  - **Stateful vs. Stateless Session Beans**

Copyright 2002, Cyberdyne Software, Inc.

Stateful vs. Stateless: Problem

- Problem:
  - Should session beans be stateful or stateless?
  - If stateless, where does the state go?
- Problem also applies to:
  - Servlets
    - Servlets must be stateless
  - SOAP/Web Services
    - Services must be stateless

Copyright 2002, Cyberdyne Software, Inc.

Stateful vs. Stateless: Considerations

- Issue:
  - When client invokes bean multiple times, client assumes bean remembers state from last time
- Stateful Session Beans
  - Stateful session beans make this easier
  - Bean per client; each client must get its bean
  - Beans must be passivated and activated
  - Beans aren't persistent
  - When can they be garbage collected?

Copyright 2002, Cyberdyne Software, Inc.

Stateful vs. Stateless: Considerations

- Stateless Session Beans
  - Better performance, scalability than stateful session beans
  - State must be stored in client, such as HttpSessionState
    - Keep state small and serializable
    - Make HTTP client identify its HttpSessionState: Cookie, URL rewriting
  - Still not persistent
    - Use persistent HTTP session support (via JDBC)

Copyright 2002, Cyberdyne Software, Inc.

Bonus Patterns: Review

- Data Transfer Object
- Identity Object
- Data Transfer Object Assembler

- Bonus Strategy:
  - Stateful vs. Stateless Session Beans

Copyright 2002, Cyberdyne Software, Inc.

Contents: Related Trends in EJB

- Layered Application Architecture
- The Patterns
- Cookbook
- Bonus Patterns
- **Related Trends in EJB**

Copyright 2002, Cyberdyne Software, Inc.

Related Trends in EJB

- EJB 2.0
  - Local Interfaces
- EJB 2.1
  - Web Service Endpoint Interfaces

Copyright 2002, Cyberdyne Software, Inc.

Local Interface

- Local Interface
  - EJB client in same JVM as EJB bean
  - Parameters passed by reference, not value
  - Parameters are not serialized
  - Methods don't throw RemoteException

Copyright 2002, Cyberdyne Software, Inc.

Local Interface: Advice

- Consequences
  - Not a Remotified Interface
  - Bean Service Interface need not be remotified
- Advice
  - Entity beans should have local interfaces only
  - Access entity beans remotely through session beans (i.e., Service Beans)

Web Service Endpoint Interface

- Web Service Endpoint Interface
  - Only applies to stateless session beans
  - Makes the bean a JAX-RPC service endpoint interface
  - Remote interface
    - Methods must declare to throw RemoteException
  - Parameters and return types must be
    - JAX-RPC types
    - Serializable

Copyright 2002, Cyberdyne Software, Inc.

Web Service Endpoint Interface: Advice

- Consequences
  - A Remotified Interface
  - Bean Service Interface must be remotified
- Advice
  - See stateful vs. stateless strategy
  - Service Bean concept still applies to web services
  - Restrictions on parameter and return types

Copyright 2002, Cyberdyne Software, Inc.

Related Trends in EJB: Review

- EJB 2.0
  - Local Interfaces
- EJB 2.1
  - Web Service Endpoint Interfaces

Contents: Review

- Layered Application Architecture
- The Patterns
- Cookbook
- Bonus Patterns
- Related Trends in EJB

Copyright 2002, Cyberdyne Software, Inc.

For More Information

- *Core J2EE Patterns* by Alur, Crupi, and Malks
- *EJB Design Patterns* by Marinescu
- *Patterns of Enterprise Application Architecture* by Fowler
- *Design Patterns* by Gamma, Helm, Johnson, Vlissides
- Sun's J2EE and EJB specifications

Copyright 2002, Cyberdyne Software, Inc.

Questions