Techniques for Building J2EE Applications

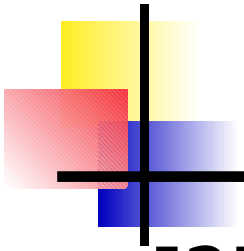
- Dave Landers
- BEA Systems, Inc.
-
- dave.landers@4dv.net
- dave.landers@bea.com

Colorado Software Summit

bea™

Why are we Here?

- Discuss issues encountered with J2EE Application deployment
- Based on my experience
  - Issues I have faced
  - Some solutions I have considered or used
- My ideas
  - Not "the" answer
  - Sometimes no answer at all
    - Just things to think about
- Goal is not to provide final answers as much as provoke thought....

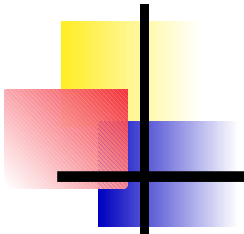


Introductions

- J2EE Developer?
  - Just EJB
  - Just WebApp
  - Applications
- Projects?
  - Large
  - Small
  - Public / Internet
  - Internal
- AppServer?
  - WebLogic
  - WebSphere
  - Orion
  - JBoss
  - Oracle
  - Other

J2EE Alphabet

- J2SE
- EJB
- JMS
- JNDI
- WebApp
- Servlet
- JSP
- JTA
- JAAS
- WebLogic
- WebSphere
- Orion
- JBoss
- Oracle
- Resin

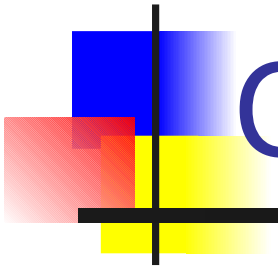


J2EE Application

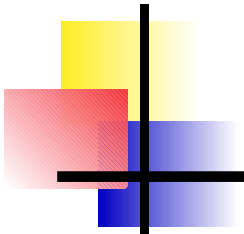
- Application – ear
  - application.xml
  - EJB - jar
    - ejb-jar.xml
  - WebApp - war
    - Servlet
    - JSP
    - TagLib
    - web.xml
  - JCA – rar

What's Left? - Today's Agenda

- Configuration
- Utility Code
  - ClassLoader Issues
  - Singletons
- Design and Deployment Issues
- Client / Server Communication
- *Etc*...
  - Vendor features *vs.* the Specs
  - Your ideas...



Configuration Issues

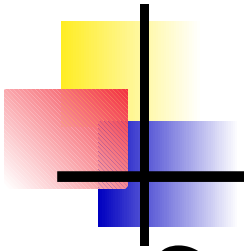


Configuration

- Resources
  - Connections to databases, *etc.*
  - <resource-ref>
- References
  - Connections to EJBs
  - <ejb-ref>
- Properties
  - Configuration parameters
  - <env-entry>

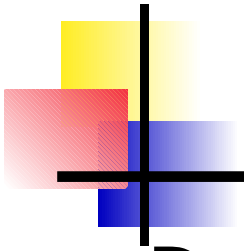
Configuration Problems

- Deployment Descriptor settings are too granular
  - Per-WebApp
  - Per-EJB (not even per module)
- What about a large deployment?
  - Dozens of EJBs in several jars, several WebApps, *etc.*
  - Too many entries to manage
  - Many required to have same settings
    - Easy to mis-configure
    - Not too bad for singly-deployed application
    - Deadly for support of "packaged" modules/applications
- Want to set in application.xml



Configuration Ideas

- Configuration should be "part of" application
  - Rather than "apart from" or "adjacent to"
- Need generalized configuration scheme
  - Separate data access (by modules) from data storage
    - Accessing configuration properties *vs.* how it is stored
    - Storage and Persistence of configuration is a problem
  - Hide the details of this problem from user
  - J2EE does this
    - JNDI for access
    - Deployment descriptor for configuration data storage
    - Not available at application level



Configuration Persistence Ideas

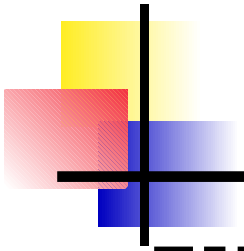
- Deployment Descriptor editor tools
  - Maybe specific to your application
  - Make bulk changes atomically
  - Natural migration if/when J2EE or vendor provides application-level settings
- Configuration Objects
  - Factory / Singleton pattern
  - Deal with configuration data storage problem
    - File or database or ... ?
      - Depends on nature of data: size, life cycle, structure, *etc.*

File-Based Configuration

- XML-structured configuration data
- File somewhere in ear
  - META-INF/foo-config.xml ?
- Can't read using "normal" File I/O
  - Where is ear? Where is file? How to do this?
- ClassLoader.getResourceAsStream() ?
  - What ClassLoader? Class-Path?
  - Not spec'd – depends on AppServer
  - More on ClassLoader later
- Plug in implementation based on AppServer, *etc.*

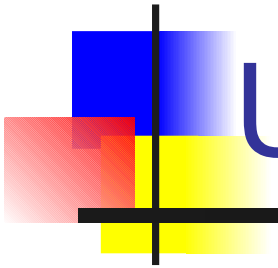
Database for Configuration

- Persistence through known, standard APIs
- Versioning, viewing, changing
  - How to check in to source control?
- Staging
  - Moving configuration from dev to stage to production?
- Can still have file-based representation
  - Write a configuration database loader
    - But this represents an extra deployment step
    - How to synchronize with redeploy?
- Configure access to the configuration database?
  - Still have potential problem setting resource-ref

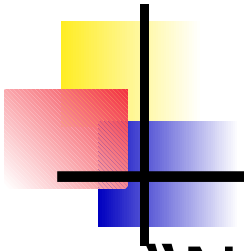


Other Configuration Ideas

- EJB as Configuration Objects
  - Entity bean mapping to database
  - Session bean aggregating common env-entry properties in one deployment descriptor
- JNDI
  - How to do initialization: bind() ?
- JMX MBeans
  - Still have initialization problem
  - Not widely adopted
  - Remote access (cluster)?
  - SNMP Management available

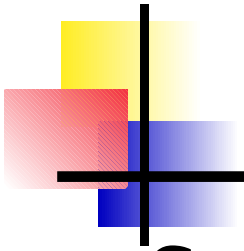


Utility Code



Utility Code

- "Normal" code that is not a Module
  - Doesn't fit EJB or JCA
  - Used by Servlet, TagLib, EJB, *etc.*
- Where to put it?
- Is it deployable?
- Shared with other modules?
- Design Pattern Issues (Singletons)



Scoping of Utilities

- Server scoped
- Application scoped
  - Available to all modules in application
  - Isolated from other applications
  - Is this possible with distributed application?
    - Not unless you use a J2EE technology
    - Application/server scoped
- Module scoped

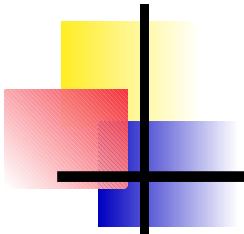
Module-Scoped Utilities

- Used only by a single module
- Easy – make it part of that module
  - In ejb-jar or rar
  - Jar in WEB-INF/lib
  - Classes in WEB-INF/classes
  - Reference it via jar's Manifest Class-Path entry
- Probably hot-deploys with the module or ear
- What about inter-module isolation?
  - Used by two modules – can they be isolated?
  - Not really defined
  - Depends on AppServer & ClassLoader architecture

Shared utilities

- Several modules need access to same class
  - Not just same code, but same class instance
  - Static methods or “Singletons” (i.e. Cache)
- Want to just put them somewhere in the ear
  - Deploy with application
  - Ear is not a “real jar”
  - No META-INF/lib or META-INF/classes
  - No <class-path> in application.xml
  - Ear’s manifest Class-Path is not used

- What to do?



Shared utilities

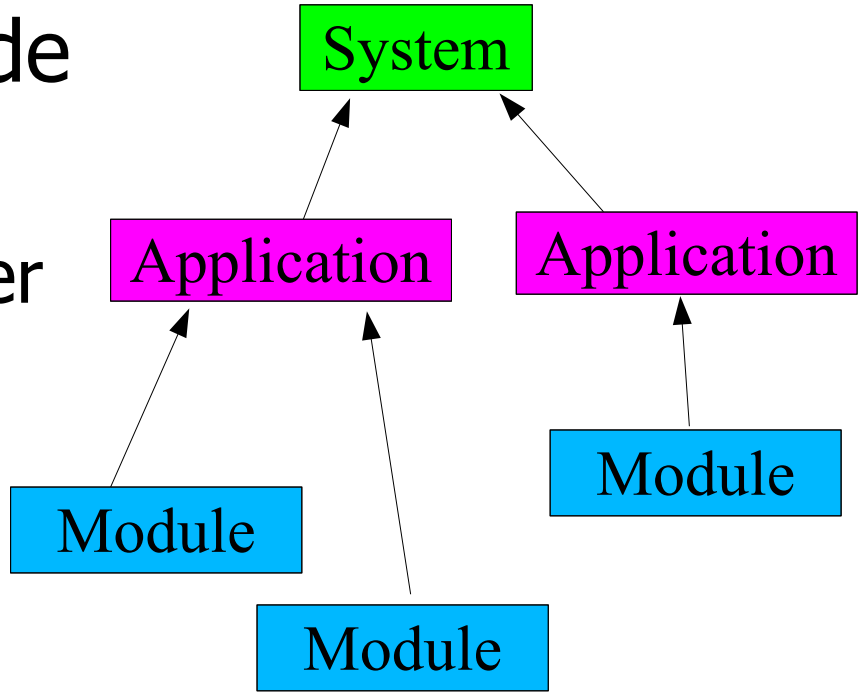
- Server's CLASSPATH
  - Works, if you don't need Application Isolation
    - All applications share same classes
  - Not "hot-deployable"
  - Not "packaged" with application
    - Use depends on server setup not just deployment
- Other solutions depend on AppServer
  - Some do load classes in ear
  - Some have ear "lib" directory
  - May depend on AppServer's ClassLoader architecture

ClassLoader Basics

- Parent ClassLoader is system "CLASSPATH"
  - Other ClassLoaders are children of system – in tree
- Loading a class:
  - Delegate to parent first, otherwise child can load it
  - Can't "see" classes loaded by siblings
    - Might have multiple instances of same class in JVM
      - (myFoo instanceof Foo) == false !
      - (Foo) myFoo ClassCastException !
- Instance of class tied to ClassLoader
  - Reload classes by "throwing away" ClassLoader
    - JSP reloads, EJB redeploy, *etc.*

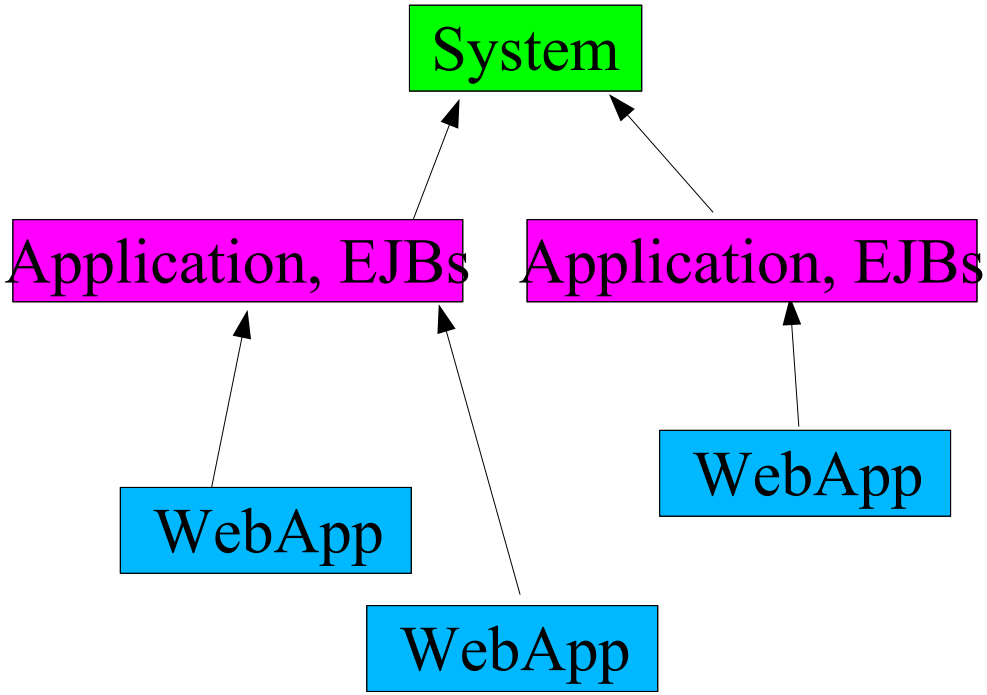
ClassLoader Architectures

- WebSphere 4.0, Module Mode
  - Three other modes
  - Shared Application ClassLoader
  - ClassLoader for each Module
    - EJB
    - WebApp
    - Manifest Class-Path entry
  - Sibling Module ClassLoaders are aware of each other by grouping of Modules
    - Details not exposed in docs



ClassLoader Architectures

- WebLogic 6.x, 7.0
  - Application ClassLoader
    - All EJBs in ear
    - Manifest Class-Path for jars and wars
  - WebApp ClassLoader
    - WEB-INF/classes
    - WEB-INF/lib
    - JSPs



ClassLoader Architectures

- Not called out in the J2EE Spec
- Differences between AppServers affect portability
  - Example:
    - fooEjb.jar references bar.jar in Manifest Class-Path
    - WebLogic: bar.jar available to al EJBs and WebApps
    - WebSphere: bar.jar only made available to fooEjb
  - Affects how you reference utility classes
    - For most AppServers, differences are probably in the deployment or assembly, not in the function
    - Better check your AppServer documentation
      - Or experiment....

What is a Singleton?

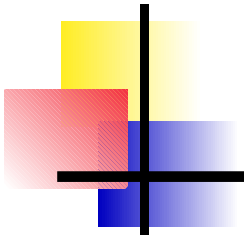
- GoF – A class with only a single instance
  - Standard code pattern using static "instance" field

```
public class SingleThing
{
    private static SingleThing INSTANCE
        = new SingleThing();

    private SingleThing()
    {
    }
    // etc ...
}
```

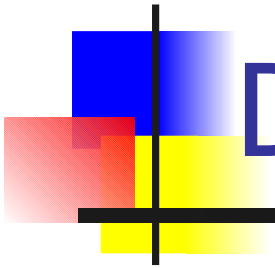
How Single is a Singleton?

- J2EE system may be spread across many machines, several JVMs, several ClassLoaders
- Static fields – per Class instance, per ClassLoader
- "Singleton" with traditional implementation is "scoped" to the ClassLoader (tree) that loaded it
  - Usually OK in J2EE (cache, resource access, *etc.*)
  - Depends on ClassLoader architecture
  - How do you get a "real" Singleton?

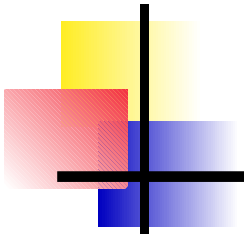


"Real" Singleton?

- SessionBean
  - Stateless – might be pooled, clustered – not single
    - Unless AppServer lets you specify pool size, pinned deployment, *etc.*
  - Stateful – tied to a client – not useful here
- EntityBean
  - Natural for Application-wide Singleton data
    - Backed by Database as single-source
  - Probably not appropriate for Application-wide Singleton services

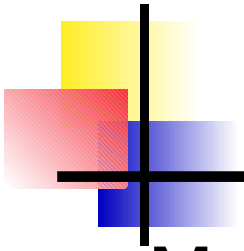


Design and Deployment



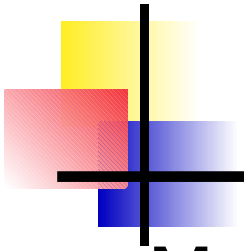
Reuse

- Modularity
  - Design
  - Deployment
- Inheritance
  - EJB, WebApp
  - Module inheritance
  - Object inheritance



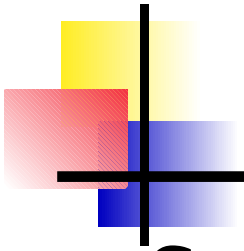
Modularity

- Multiple Modules and Applications
  - Multiple deployment of same Module in Application
    - Same EJB deployed twice with different configuration
    - Two catalogs, different databases, *etc.*
  - Multiple deployments of same Application
    - Maybe sharing some resources
      - At least sharing the AppServer as a resource
    - Different configurations
    - Multiple storefronts, Internal/External portals, *etc.*
  - Isolation
  - Configuration
  - Assembly



Reuse

- Modules should be reusable
  - Design for reuse
- What about interdependent modules
  - ejb-ref helps define dependencies on EJBs
  - ear helps make a package
  - Would be nice to have “aggregate” module archive
    - ear-within-an-ear



Multiple Deployments

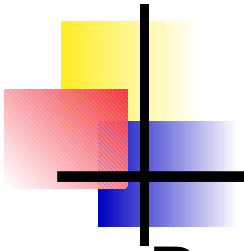
- Same Module, deployed to two Applications?
  - Do settings in deployment descriptors conflict?
    - i.e. global JNDI names – bind error
      - In vendor descriptor
      - Leave unbound, and use ejb-link (if AppServer allows)
  - Externalize configuration settings
    - To deployment descriptor or application-scoped file
    - Document the configuration options / settings

Module Inheritance

- Inheritance
  - Primary mechanism of reuse
  - EJB has deferred this in every spec since 1.0 – shame!
  - EJBHome is not a real Factory
    - Type specific to Component (Remote/Local) interface
      - AnimalHome must return Animal
      - Can't retrieve Dog, Cat, Sheep instances
      - Client can't cast to Sheep and call baahhh()
  - Polymorphism has to be done in the EnterpriseBean (implementation) class by delegation
      - Rather than at module level
    - Especially difficult with Entity (CMP)

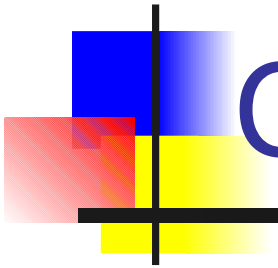
Local / Remote Transparency

- EJBHome is specific to Local or Remote
  - Often would like to use "best choice"
    - When allowed, available, desired
    - Allows split deployments (separate servers) when appropriate
    - Better performance when co-located
      - Pass-by-value means no Serialization cost
      - Some AppServers allow this optimization on co-located Remotes
    - Allow assembly-time decision rather than development-time
  - No common base interface for Home or Component interfaces
    - Signature differences required by spec
  - RemoteException is checked exception

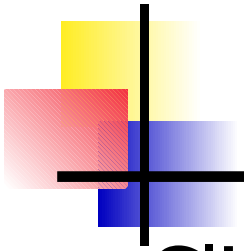


WebApp "Inheritance"

- Base WebApp with core features
- Customized with Application-specific additions or changes
  - Sounds like Polymorphism of a sort
- Solutions?
  - Build-time ("file flinging")
  - Would like "war merging" deployment feature

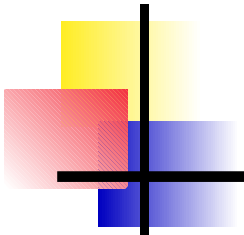


Client / Server Interactions



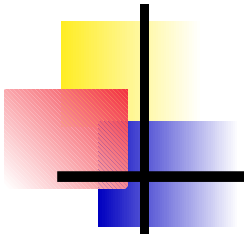
Clients

- Client is way user interacts with Application
- Browser Based
  - Implemented in Server (JSP,HTML, *etc.*)
  - Limited functionality
  - Easy to manage for thousands of clients
- Programmatic Clients
  - Touch EJBs, Servlets, *etc.* directly
  - Broader functionality possible
  - Management is a problem for large client base



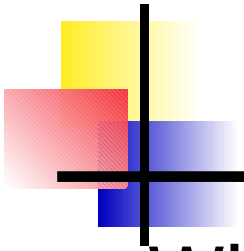
Client Access

- Browser clients – easy
  - Code on server, under your control
  - Can be packaged as part of application – war in ear
- Programmatic clients
  - Connection issues
    - Firewall, *etc.*
  - Deploy client interfaces
    - Managing updates
      - Remote interfaces and utilities and client code
      - URLClassLoader
      - Java Web Start?



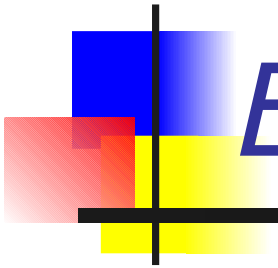
Client Access

- How about HTTP as transport?
  - Rather than JNDI/RMI/CORBA
  - Reduce "contact area" of client with server
    - Reduce number of javax.* packages to import
    - Open up for non-Java clients
      - Perl, python, *etc.* all understand HTTP
  - Custom Servlet to dispatch client requests
    - Serialized Objects as protocol?
      - Specific to Java
    - Custom XML protocol?
      - Open



Client Access

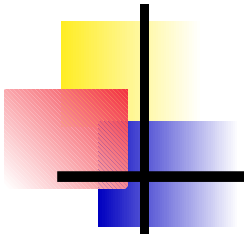
- What about WebService as "API"
  - JAX-RPC on client
    - Or URLConnection and XML Parser
  - Lighter weight, no "ejb client jar"
    - Still have JAX-RPC proxy classes
  - Still have to manage updates to client code
  - Remote interface is WSDL (and Proxies)
  - Security? (https? WS-Security? SAML?)
  - Transactions? (at EJB? WS-Transacton?)
  - State
    - WS is stateless, not conversational



Etc....

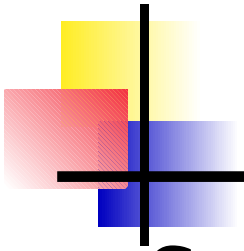
Standards *vs.* Vendor Features

- AppServer Features
  - Locks you in unless you abstract behind pluggable API
  - Might not be API feature, but behavior or configuration
    - Architecture of ClassLoaders, Threads, *etc.*
  - Standards outside J2EE
    - JMX, JAX*, Draft spec previews, WebServices
  - Non-Standard but useful things
    - Extensions to the specs
    - EJB-QL enhancements
    - Performance enhancements (pass by value *vs.* by reference)
- Bottom line – Happy customer (working code)



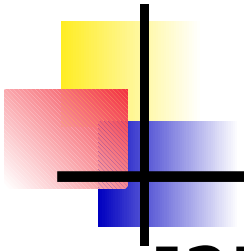
Build Structure

- Code Categories
  - Reusable Module code
    - EJBs, WebApps, TagLibs, utilities
    - Usable by more than one application
  - Application-specific code
    - EJBs, WebApps, JSPs
    - Specific to a single application
  - Build each as you would any other source
    - Application build depends on modules
      - Copies ejb-jars, WebApps, TagLibs into ear



Build Structure

- Separate source directory for each jar
  - Can control code dependencies and class path
- Keep build output separate from source
  - Easier clean, don't confuse source control
- Compile EJBs and JSPs (ejbc, jspc)
  - Javac can't check EJB compliance (naming patterns)
  - JSP is source, too – compile it
    - Catch problems without having to browse to each page
- Might want server "configuration build"
- Source control!
- Ant!

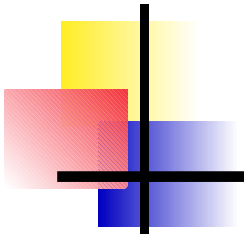


Your Turn . . .

- J2EE Problems? Annoyances? Praise?
- Solutions you have used?
  - The Good, the Bad, and the Ugly
- Recommendations?
- Do you care about Vendor Specific Features?

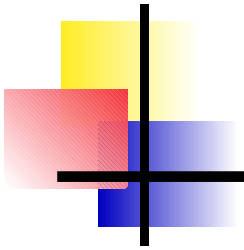
Web References

- http://java.sun.com/j2ee
- EJB Spec
  - http://java.sun.com/products/ejb/docs.html
- WebApp / Servlet / JSP Spec
  - http://java.sun.com/products/servlet/download.html
  - http://java.sun.com/products/jsp/download.html
- J2EE Spec, Blueprints, *etc.*
  - http://java.sun.com/j2ee/download.html
- J2EE API Docs
  - http://java.sun.com/j2ee/sdk_1.3/techdocs/api



Web References

- The Server Side
  - http://www.theserverside.com
  - News, Patterns, Discussion, Downloads, *etc.*
- ONJava
  - http://www.onjava.com
  - O'Reilly – Articles, etc.
- Your AppServer documentation



The End – Thank You

Please fill out evaluations

dave.landers@4dv.net

dave.landers@bea.com