ENTERPRISE JAVABEANS™ (EJB™) 3.1 TECHNOLOGY

Kenneth Saks
Senior Staff Engineer
SUN Microsystems

TS-5343

Java

Sun microsystems

Learn what is planned for the next version of Enterprise JavaBeans™ (EJB™) technology

GOAL

Agenda

- Introduction
- Proposed Functionality
- Summary
- Q & A

EJB 3.0 Specification (JSR 220)

- Features
  - Simplified EJB API
  - Java Platform Persistence API
- Focus on ease-of-use
  - Annotations
  - Optional ejb-jar.xml
  - Intelligent defaults
  - Fewer classes needed
- Very well received within the community
  - But... more work is needed

EJB 3.1 Specification

- Goals
  - Continued focus on ease-of-use
  - New features
- JSR (Java Specification Request) 318
  - Launched in August 2007
    - Diverse 20 member expert group
    - Java Persistence API will evolve separately (JSR 317)
  - Early Draft released in February 2008
  - Part of Java Platform, Enterprise Edition (Java EE Platform) 6
- Caveat – Work In Progress
  - Features / APIs still subject to change

Agenda

- **Introduction**
- **Proposed Functionality**
- Summary
- Q & A

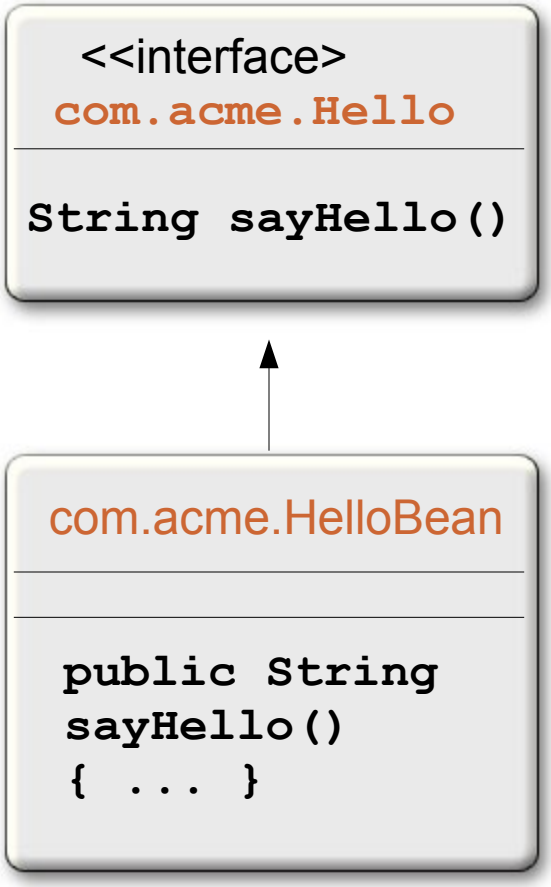
Ease of Use Improvements

- Optional Local Business Interfaces
- Simplified Packaging
- EJB Technology “Lite”
- Portable Global JNDI Names
- Simplified EJB Component Testing

Session Bean with Local Business Interface

HelloBean Client

```
@EJB
private Hello h;

...

h.sayHello();
```



Optional Local Business Interfaces

- Sometimes separate local business interface isn't needed
- Better to completely remove interface from developer's view than to generate it
- *Result : "no-interface" view*
  - *Just a bean class*
  - *All public bean class methods exposed to client*
  - *Same behavior and client programming model as Local view*
    - *Client still acquires an EJB specification reference instead of calling* `new()`

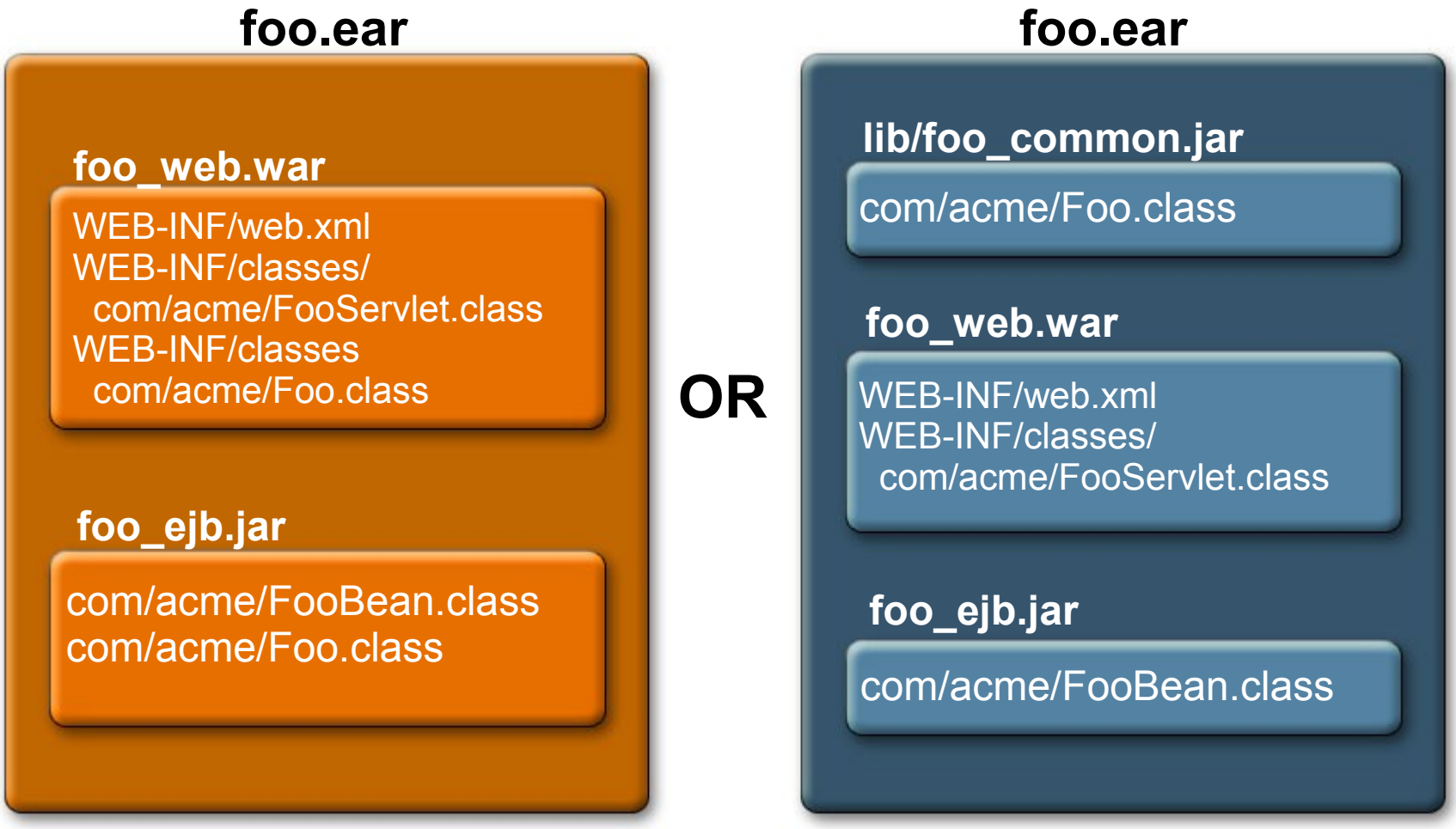
Session Bean with “No-interface” View

```
@Stateless
public class HelloBean {

    public String sayHello(String msg) {
        return “Hello “ + msg;
    }

}
```

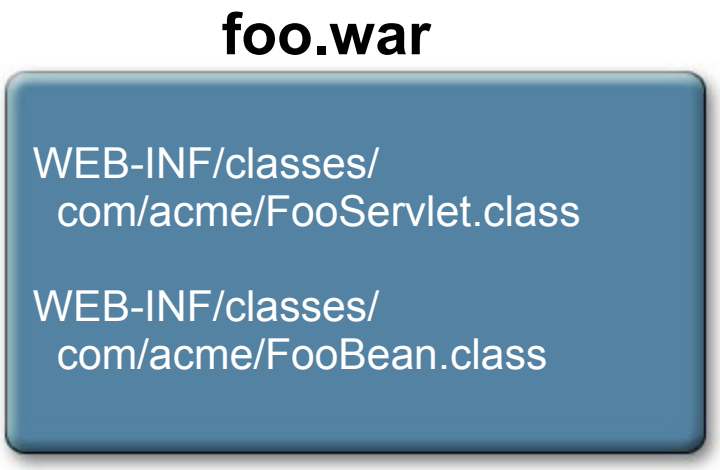
No-interface View Client

```
@EJB HelloBean h;

...

h.sayHello("bob");
```

Web/EJB Technology Application in Java™ EE Platform 5



Web/EJB Technology Application in Java™ EE Platform 6



Simplified Packaging

- Goal is to remove an artificial packaging restriction
  - NOT to create a new flavor of EJB component
- EJB component behavior is independent of packaging
  - One exception : module-level vs. component-level environment
- Full EJB container functionality available

EJB Technology "Lite"

- Small subset of EJB 3.1 API for use in Web Profile
- Broaden the availability of EJB technology
  - Without losing portability
- Same exact Lite application can be deployed to Web Profile and Full Profile
  - Thanks to simplified .war packaging
- Open issue : whether Web Profile will **require** Lite API

“Lite” vs. Full Functionality

Lite

- Local Session Beans
- Annotations / ejb-jar.xml
- CMT / BMT
- Declarative Security
- Interceptors

- (Also requires JPA 2.0 API / JTA 1.1 API)

Full = Lite + the following:

- Message Driven Beans
- EJB Component Web Service Endpoints
- RMI-IIOP Interoperability
- 2.x / 3.x Remote view
- 2.x Local view
- Timer Service
- CMP / BMP

Session Bean Exposing a Remote View

```
@Stateless
@Remote(Hello.class)
public class HelloBean implements Hello {

    public String sayHello(String msg) {
        return "Hello " + msg;
    }

}
```

Remote Clients

```
// Client in a Java EE container
@EJB Hello hello;


// Client in a Java SE environment
Hello hello = (Hello)
  new InitialContext().lookup(???);
```

Question : How does the caller find the target EJB component?

Problems with “Global” JNDI Names

- Not portable
  - Global JNDI namespace is not defined in Java EE platform specification
  - Vendor-specific configuration needed for each deployment
- No standard syntax
  - Can they contain : “.”, “_”, “/” ?
- Not clear which resources have them
  - Local EJB interfaces?
- “mappedName” is only a partial solution
- Not unique to EJB components: Datasources, Queues, etc.

We're investigating ways to fix these issues. Stay tuned...

EJB Component Testing

- It's too hard to test EJB components, especially Local
  - Forced to go through Remote facade or Web tier
  - Separate JVM™ software processes needed for server and client
- Support for client-side EJB component execution exists, but...
  - Not present in all EJB Technology implementations
  - No standard behavior for bootstrapping, component discovery, shutdown etc.

Example: Local Stateless Session Bean

```
@Stateless
@Local(Bank.class)
public class BankBean implements Bank {

  @PersistenceContext EntityManager accountDB;

  public String createAccount(AccountDetails d) {
    ...
  }

  public void removeAccount(String accountID) {
    ...
  }

}
```

Example: Test Client

```
public class BankTester {

  public static void main(String[] args) {

    // Possible container bootstrapping API
    EJBContainer container = EJBContainer.init();

    // Acquire Local EJB component reference
    Bank bank = (Bank)
      container.getContext().lookup("BankBean");

    testAccountCreation(bank);
    testAccountRemoval(bank);

  }
```

Example: Test Client Execution

```
% java -classpath bankClient.jar :
                  bankEjb.jar :
                  javaee.jar :
                  <vendor_rt>.jar

                  com.acme.BankTester
```

Client-side EJB Component Execution

- Support Java SE platform clients
- “Single application” model
- Same EJB component behavior / life cycle as server-side
  - CMT/BMT, injection, threading guarantees, etc.
- Still TBD:
  - Bootstrapping
    - Define new API or add properties to InitialContext() ?
  - Component discovery rules
  - Injection into client test code
  - Entire 3.1 API or Lite only ?

New Features

- Singletons
- Application startup / shutdown callbacks
- Calendar-based timer expressions
- Automatic timer creation
- Simple Asynchrony

Singletons

- New session bean component type
  - Provides easy sharing of state within application
  - One singleton bean instance per application per JVM version
    - Once created, lives for the duration of its container
  - Instance state is not preserved after container shutdown
- Lots in common with stateless / stateful beans
  - Client views (No-interface , Local, Remote, Web Service)
  - CMT / BMT
  - Container services: resource managers, timer service, etc.

Simple Singleton

```
@Singleton
public class SharedBean {

  private SharedData shared;

  @PostConstruct private void init() {
    shared = ...;
  }

  public int getXYZ() {
    return shared.xyz;
  }

}
```

Singleton Client

```
@Stateless
public class FooBean {

  // Inject reference to Singleton bean
  @EJB
  private SharedBean shared;

  public void foo() {
    int xyz = shared.getXYZ();
    ...
  }

}
```

Singleton Concurrency Options

- Single threaded (default)
  - Container serializes concurrent requests
- Container Managed Concurrency
  - Concurrency via method-level locking metadata
    - Shared lock: allow any number of concurrent accesses
    - Exclusive lock: ensure single-threaded access
  - Container blocks invocations until they can proceed
    - ...or until app-specified timeout is reached
- Bean Managed Concurrency
  - Like Java Servlet API threading model
  - Application can use Java SE Platform synchronization primitives

Read-Only Singleton with Container Managed Concurrency

```
@Singleton
public class SharedBean {

  private SharedData shared;

  @ReadOnly
  public int getXYZ() {
    return shared.xyz;
  }

  @PostConstruct void init() {
    shared = ...;
  }
}
```

Read-Mostly Singleton with Container Managed Concurrency

```
@Singleton
@ReadOnly
public class SharedBean {

  private SharedData shared;

  public int getXYZ() {
    return shared.xyz;
  }

  @ReadWrite
  public void update(...) {
    // update shared data
    ...
  }
```

Concurrent Access Timeouts

```
@Singleton
@ReadOnly
public class SharedBean {

  private SharedData shared;

  @AccessTimeout(1000)
  public int getXYZ() {
    return shared.xyz;
  }

  @ReadWrite
  public void update(...) {
    // update shared data
  }
```

Read-Mostly Singleton with Bean Managed Concurrency

```
@Singleton
@BeanManagedConcurrency
public class SharedBean {

  private SharedData shared;

  synchronized public int getXYZ() {
    return shared.xyz;
  }

  synchronized public void update(...) {
    // update shared data
    ...
  }
```

Startup / Shutdown Callbacks

- Run code during application initialization or shutdown
- Callbacks defined on Singleton classes
- Full container services available within callbacks
  - Other EJB components
  - Timer Service
  - Resource Managers (e.g. Entity Managers)
- @Startup
  - Forces eager singleton initialization
  - Combine with @PostConstruct for initialization-time callback
- @PreDestroy
  - Called during application shutdown
  - Independent of eager/lazy initialization

Startup / Shutdown Callbacks

```
@Singleton
@Startup
public class StartupBean {

  @PostConstruct
  private void onStartup() {
    ...
  }

  @PreDestroy
  private void onShutdown() {
    ...
  }
```

Timer Service Features

- Calendar-based timeouts
- Automatic timer creation

Calendar Based Timeouts

- Cron semantics with improved syntax
- Usable with programmatic or automatic timer creation
- Named attributes
  - second, minute, hour
    - Default = “0”
  - dayOfMonth, month, dayOfWeek, year
    - Default = “*”
- Attribute syntax
  - Single value : minute = “30”
  - List : month = “Jan, Jul, Dec”
  - Range : dayOfWeek = “Mon-Fri”
  - Wildcard : hour = “*”
  - Others ???

Programmatic Calendar-Based Timer

```
@Stateless
public class LunchNotificationBean {

  @Resource TimerService timerSvc;

  public void createLunchNotification() {
    // Every weekday at noon
    ScheduleExpression expr = new ScheduleExpression().
      dayOfWeek("Mon-Fri").hour(12);
    timerSvc.createTimer(expr, null);
  }

  @Timeout void sendLunchNotification(Timer t) {
    ...
  }

}
```

Automatic Timer Creation

- Container creates timer automatically at deployment time
- Logically equivalent to one createTimer() invocation
- Differences between automatic and programmatic timers:
  - No “info” object
  - Each automatic timer can have its own timeout callback method

Automatic Timer Creation

```
@Stateless
public class BankBean {

  @PersistenceContext EntityManager accountDB;
  @Resource javax.mail.Session mailSession;

  // Callback the 1st of each month at 8 a.m.

  @Schedule(hour="8", dayOfMonth="1")
  void sendMonthlyBankStatements() {
    ...
  }

}
```

Simple Asynchrony

- Different styles
  - Local concurrency as code composition mechanism
    - E.g : break large piece of work into independent tasks
  - Async RPC
  - Messaging between tightly coupled components
    - Transacted send / guaranteed delivery , without need for loose-coupling
- Too difficult with existing APIs
  - JMS API – complex API / lots of configuration
  - Timer Service – different design center
  - Threads – not well integrated with component model
- Approach : integrate asynchronous support directly into session bean components

Simple Local Concurrent Computation

```
@Stateless public class DocBean {

  @Resource SessionContext ctx;
  @PersistenceContext EntityManager resultsDB;

  public void processDocument(Document document) {
    DocBean me = ctx.getBusinessObject(DocBean.class);
    me.doAnalysisA(document);
    me.doAnalysisB(document);
  }

  @Asynchronous public void doAnalysisA(Document d) {...}

  @Asynchronous public void doAnalysisB(Document d) {...}

}
```

Asynchronous Operation Results

- Based on java.util.concurrent.Future<V>
- For
  - Result values
  - System/Application exceptions thrown from target bean
  - Cancelation requests
- Method signature declares return type as Future<V>
  - e.g. public Future<int> compute(Task t)
- Method implementation returns javax.ejb.AsyncResult<V>
  - Concrete helper class for passing result value to container
  - javax.ejb.AsyncResult<V> implements Future<V>
  - Constructor takes result value as argument

Asynchronous Operation Results -- Client

```
@EJB Processor processor;

Task task = new Task(...);

Future<int> computeTask = processor.compute(task);

...

int result = computeTask.get();
```

Asynchronous Operation Results

```
@Stateless
public class ProcessorBean implements Processor {

  @PersistenceContext EntityManager db;

  @Asynchronous
  public Future<int> compute(Task t) {

    // perform computation
    int result = ...;

    return new javax.ejb.AsyncResult<int>(result);
  }

}
```

Canceling Long Running Operations

```
@Stateless public class ProcessorBean implements Processor
{
  @Resource SessionContext ctx;
  @PersistenceContext EntityManager db;

  @Asynchronous public Future<int> compute(Task t)
    throws CanceledException {

    ...

    if(ctx.isCanceled()) {
      ctx.setRollbackOnly(true);
      throw new CanceledException();
    }

    int result = ...;
    return new javax.ejb.AsyncResult<int>(result);
  }
```

Async Behavior

- Transactions
  - No transaction propagation from caller to callee
- Method authorization
  - Works the same as for synchronous invocations
- Exceptions
  - Exception thrown from target invocation available via Future<V>.get()
- Transacted send (Optional)
  - Forward progress of async invocation tied to client tx
- Persistent delivery (Optional)
  - Target component invocation is guaranteed to occur
  - Requires Serializable parameter types

Business Process Transition

```
@Stateless
public class Step1Bean implements Step1 {

  @EJB Step2 step2;
  @PersistenceContext EntityManager database;

  public void do(BusinessProcess b) {

    ...

    // Transition to next step upon transaction commit
    step2.do(b);

  }
}
```

Business Process Transition

```
@Stateless
@Remote(Step2.class)
public class Step2Bean implements Step2 {

  @PersistenceContext EntityManager database;

  @Asynchronous(transactedSend=true, persistent=true)
  public void do(BusinessProcess b) {

    ...

  }

}
```

Agenda

- Introduction
- Proposed Functionality
- Summary
- Q & A

Summary

- EJB 3.1 Specification (JSR 318)
- Part of Java EE™ Platform 6
- Goals
  - Ease-of-use
  - New Features

For More Information

- EJB 3.1 Technology Community Discussion
  - BOF 5753
  - Wednesday 6:30 – 7:20
- JSR 318 Home : http://jcp.org/en/jsr/detail?id=318
  - Send comments to jsr-318-comments@jcp.org
- Blog : http://blogs.sun.com/kensaks/
- Reference Implementation : GlassFish project V3
  - http://glassfish.dev.java.net
  - ejb@glassfish.dev.java.net

Agenda

- Introduction
- Proposed Functionality
- Summary
- Q & A

THANK YOU

Kenneth Saks
kenneth.saks@sun.com

TS-5343