Enterprise JavaBeans™ 3.1 Technology

Kenneth Saks

Senior Staff Engineer

Sun Microsystems

TS-4247

Objective

Learn about the new features planned for Enterprise JavaBeans™ (EJB™) technology.

Agenda

Overview

Ease of Use Enhancements

New Features

Summary

Q&A

EJB 3.0 Specification (JSR 220)

- Final Release May 2006
  - Part of Java™ Platform, Enterprise Edition (Java™ EE Platform) 5
- Features
  - Simplified EJB API
  - Java Persistence API
- Approach
  - POJO style development
  - Leverage Java Platform, Standard Edition (Java SE Platform) 5 annotations
  - Minimize use of XML deployment descriptors
  - Intelligent defaults

JSR = Java Specification Request

EJB 3.1 Technology Motivation

- Further improve ease-of-use
  - Reduce number of required interfaces
  - Loosen packaging restrictions
- Add features that could not be realized in EJB 3.0 specification
  - Since earlier focus was ease-of-use

EJB 3.1 Technology

- Part of Java EE platform 6
- Scope is EJB components only
  - Java Persistence API will evolve as a separate specification and expert group
- Timeline
  - Submit JSR: May 2007
  - Community Review: August 2007
  - Public Review: January 2008
  - Final Release: Q3 2008

Agenda

Overview

Ease of Use Enhancements

New Features

Summary

Q&A

Ease of Use Enhancements

- Optional Local business interfaces
  - Develop local EJB components using only a bean class
- EJB components in the web tier
  - Package/deploy EJB components in a .war without an ejb-jar

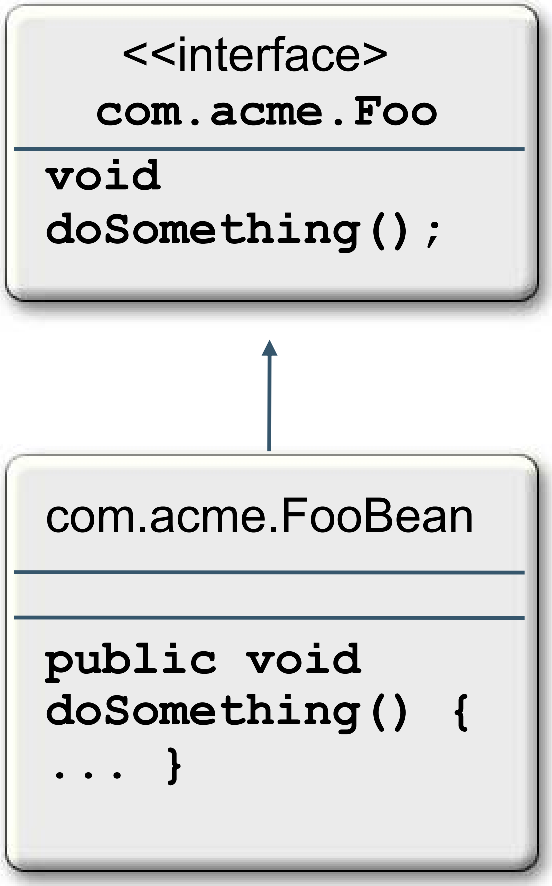
Session Bean with Local Business Interface

FooBean Local Client

```
// inject ejb ref
@EJB
private Foo foo;


...

// call FooBean
foo.doSomething()
;
```



```
<<interface>
com.acme.Foo
void
doSomething();
```

```
com.acme.FooBean

public void
doSomething() {
... }
```

EJB 3.0 Technology Local Client Programming Model

- Define a dependency on a Local EJB component
  - Via annotation(@EJB) or XML(ejb-local-ref)
  - Dependency Type is <local business interface>
- Inject or lookup the dependency to acquire a reference object
  - Client **never** calls **new()** on <bean class>
- Local reference is a special container object, **not** a bean instance
- Caller may invoke any methods defined on Local business interface, but cannot directly access bean instance state

EJB 3.0 Technology Local Client Programming Model (Cont.)

- Separation of client reference and bean instance allows container to provide:
  - Efficient resource management
    - Pooling of stateless session bean instances
    - Activation/Passivation of stateful session beans
  - Lazy initialization
  - Transparent clustering support
  - Concurrency control
    - Single-threaded bean instance guarantee without use of Java SE platform level synchronization

Local Business Interfaces

- In some cases, separating Local business interface and bean class does not add much value
  - Local EJB components often invoked through an expression language
  - Local EJB components/clients packaged in same application
    - Same class loader
  - Local EJB components often already very fine grained and tightly coupled to Local client
  - Very rare to provide different bean implementations for same Local business interface
- If not strictly needed, only adds to development/ maintenance burden

Optional Local Business Interfaces

- Make Local business interface **optional**
- But…preserve separation between client reference objects and bean instances
  - Client still **never** uses new() to obtain a reference
- Reference is of type <bean class> but client contract only exposes the EJB component's public Local business methods
- Local client programming model essentially the same with/without Local business interface

Local Session Bean Without Business Interface

```
@Stateless public class FooBean {

   // Local business method doSomething()
   public void doSomething() { ... }

}
```

Client of Local Session Bean Without Business Interface

```
@Stateless public class BarBean implements BarRemote {

    // Inject ejb reference to FooBean
    @EJB FooBean foo;

    public void businessMethod() {

        // WRONG. Even though Bean has no local business
        // interface, client does not use new()
    // foo = new FooBean();

        // Call FooBean local business method
        foo.doSomething();

}
}
```

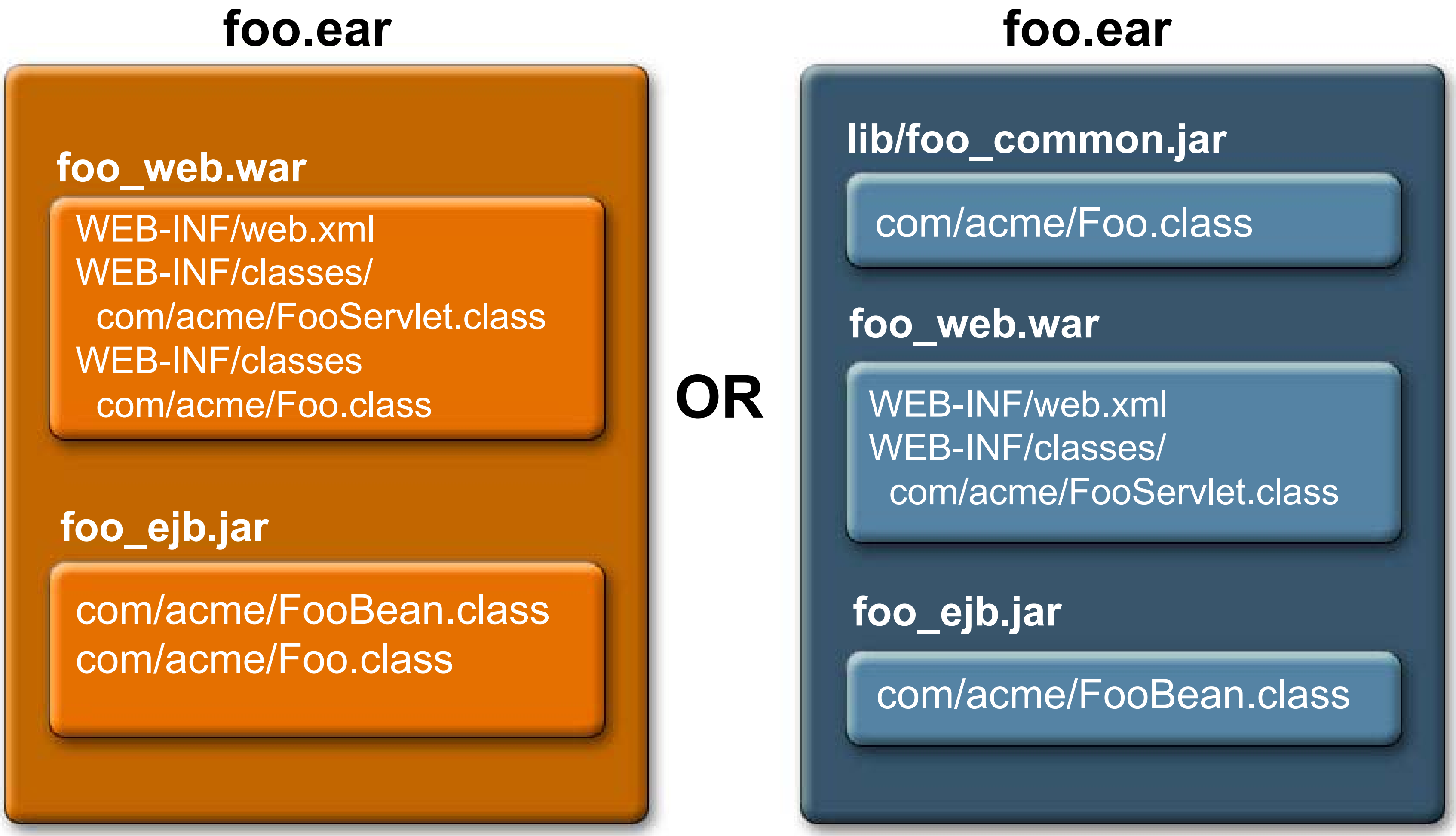
Optional Local Business Interfaces

- Further simplifies development of Local EJB components
  - Less code to write/package/maintain
  - Completely removing an interface from developer view gives biggest ease-of-use improvement
    - Better than relying on IDEs to generate interface and keep it in sync
- Does not introduce significant incremental coupling
- Easy transition from earlier Local client view
- Optional—Local business interfaces still fully supported

EJB Component Usage From Web Tier

- Mostly accessing EJB components within same application
- Mostly using Local Stateless/Stateful session beans
- Simplified API has increased the usage of EJB technology from the web tier
- More simplifications needed, especially with packaging requirements

Combined Web/EJB Technology Application in Java EE Platform 5



foo.ear

foo_web.war

```
WEB-INF/web.xml
WEB-INF/classes/
  com/acme/FooServlet.class
WEB-INF/classes
  com/acme/Foo.class
```

foo_ejb.jar

```
com/acme/FooBean.class
com/acme/Foo.class
```

OR

foo.ear

lib/foo_common.jar

```
com/acme/Foo.class
```

foo_web.war

```
WEB-INF/web.xml
WEB-INF/classes/
  com/acme/FooServlet.class
```

foo_ejb.jar

```
com/acme/FooBean.class
```

Common Issues With Combined Web/EJB Technology Applications

- Requiring separate ejb-jar increases development burden/learning curve
  - ejb-jar layout/packaging different than .war
- Requires .ear module to contain .war and ejb-jar
- Confusion about how to package shared classes
  - e.g., bean interfaces, utility classes
- No sharing of component environment namespaces

Define EJB Components Within .war

foo.war

```
WEB-INF/classes/
  com/acme/FooServlet.class
WEB-INF/classes/
  com/acme/FooBean.class
```

Although most useful for Local Session Beans accessed by web components

- No ejb-jar needed
- Bean/interface/supporting classes placed in WEB-INF/classes
- One component environment (java:comp/env) shared between web application and EJB components
- Any Java Persistence API persistence units in .war are shared by EJB components
- Full EJB container functionality available
  - Although most useful for Local Session Beans accessed by web components

Define EJB Components in .war (Cont.)

- EJB components in .war have no special knowledge of web container
- Invocation semantics the same regardless of packaging
  - Transaction/Security/PersistenceContext propagation, exception behavior, etc.
- Ensures packaging decisions can be changed with minimal impact on application
  - e.g., .war becomes too big so some EJB components are moved out into separate ejb-jar

We've Come a Long Way…

J2EE Platform 1.4
foo.ear

```
META-INF/application.xml
foo_web.war
    WEB-INF/web.xml
    WEB-INF/classes
      com/acme/FooServlet.class
    WEB-INF/classes/
      com/acme/FooLocalHome.class
    WEB-INF/classes/
      com/acme/Foo.class
foo_ejb.jar
    META-INF/ejb-jar.xml
    com/acme/FooBean.class
    com/acme/Foo.class
    com/acme/FooLocalHome.class
```

Java EE Platform 6
foo.war

```
WEB-INF/classes/
  com/acme/FooServlet.class
WEB-INF/classes/
  com/acme/FooBean.class
```

Agenda

Overview

Ease of Use Enhancements

New Features

Summary

Q&A

Features

- Singleton Beans
- Additional Concurrency Options
- Timer Service Enhancements
- Simple Asynchrony
- Stateful Web Service Endpoints

State in EJB Components

- Stateful Session Beans
  - Hold ***client-specific*** state
  - Not intended to be shared by multiple clients
- Stateless Session Beans
  - No client-specific state
  - *Can* hold client-independent instance state
    - e.g., `@PersistenceContext EntityManager em;`
  - Multiple instances per bean
    - No guarantee that multiple client invocations on same bean will executed by same bean **instance**

What About Shared State?

- Very common to have state that needs to be shared across multiple components in an application
- Handled in web tier through Web Application level context (ServletContext)
  - One ServletContext per .war per server instance
  - Only accessible to associated web application
- How can state be shared between EJB components in an application?

Alternative 1: Use Stateless Session Bean Instance State

```
public class SharedData { ... }

@Stateless public class FooBean implements Foo {

    private SharedData shared;

    @PostConstruct void init() {
        // Initialize shared data
        shared = ...;
    }

    public void doSomething() {
        // Access shared state
        ...
    }
}
```

Alternative 1: Use Stateless Session Bean Instance State

- Doesn't work for mutable shared state
  - Shared state will be replicated for every bean instance created by container
  - No way for application to update each instance
- Even if state is immutable, wasteful to replicate across all instances of the bean
  - Shared state commonly used for large in-memory data structures
    - Large initialization time
    - Large memory footprint

Alternative 2: Use Stateless Session Bean Class-Level (Static) State

```
@Stateless public class FooBean implements Foo {

    static private SharedData shared;

    @PostConstruct void init() {
       synchronized(shared) {
         if( shared == null ) {
          // Initialize shared data
           shared = ...;
         }
       }
    }

    public void doSomething() { ... }
}
```

Alternative 2: Use Stateless Session Bean Class-Level (Static) State

- Behavior is too closely coupled to classloaders
    - 1 instance per classloader is **not** necessarily the same thing as 1 per Stateless Session Bean
- Container is unaware of shared state
    - No opportunity to provide value-adds
        - Additional concurrency options
        - Container initialization time callbacks

Singleton Beans

- For each singleton bean, one instance per application per server Java Virtual Machine (JVM™)
  - Not intended to provide a cluster-wide singleton
- Fits easily into existing environment dependency architecture
  - Acquisition through **`@EJB`** or lookup
  - Good for sharing data within ***entire*** application, not just EJB components
- Singleton also provides useful way to add new lifecycle callbacks for:
  - Container initialization
  - Container shutdown

The terms "Java Virtual Machine" and "JVM" mean a Virtual Machine for the Java™ platform.

Singleton

```
@Singleton
public class SharedBean implements Shared  {

   private SharedApplicationData shared;

   // Called once at container-initialization time
   @PostConstruct void init() {
       // Initialize shared data
       shared = ...;
   }

   public int getFoo() {
       return shared.foo;
   }
}
```

SLSB Client Accessing Shared State

```
@Stateless
public class FooBean implements Foo  {

    // Declare ejb dependency on Singleton bean
    @EJB private Shared shared;

    public void doSomething() {

        // Access shared data
        int foo = shared.getFoo();

        ...
    }
}
```

EJB Concurrency

- EJB Container provides single-threaded guarantee for **all** bean instances
  - SLSBs/MDBs
    - Each client invocation/message handled by different bean instance
  - SFSBs
    - Each stateful session bean identity allows one invocation at a time
- Frees bean programmer from dealing with instance state synchronization issues
  - Non-final static variables prohibited

Stateful Session Bean Client Behavior

- If a request arrives for a SFSB while it is still processing an earlier request…
  - Spec allows container to ***either:***
    - Throw `ConcurrentAccessException`
    - Serialize second request
- Developer should be able to specify desired behavior through standard metadata

```
@ConcurrencyManagement(policy=RejectConcurrentRequests)
```

Singletons and Concurrency

- Single-threaded policy too restrictive for Singletons
  - One Instance
  - Multiple clients
  - Typically read-only or read-mostly
- Need new concurrent access options to allow for balance of performance vs. code complexity
  - Container-managed concurrency
    - Method-level locking metadata
  - Bean-managed concurrency
    - Direct use of `synchronized`
  - Allow for Singletons only or all component types?

Singleton With Immutable Data

```
@Singleton @ReadOnly
public class SharedBean implements Shared {

    private SharedApplicationData shared;

    // Called once at container-initialization time
    @PostConstruct void init() {
        // Initialize shared data
        shared = ...;
    }

    public int getFoo() {
        return shared.foo;
    }
}
```

Singleton With Some Updates

```
@Singleton
public class SharedBean implements Shared  {

    private SharedApplicationData shared;

    @ReadOnly public int getFoo() {
        return shared.foo;
    }

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


    ...
}
```

Concurrency Fully Controlled by Bean

```
@Singleton
@BeanManagedConcurrency
public class SharedBean implements Shared  {

   private SharedApplicationData shared;

   synchronized public int getFoo() { return shared.foo; }

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

   ...
}
```

Concurrency Recap

- No change to default concurrency behavior for existing component types
- Allow specification of client behavior for concurrent attempts to access non-shared SFSBs
- For shared bean instances, favor container-managed concurrency
  - Allows container to define concurrency semantics
  - Use of annotations allows for flexible configuration
    - e.g., class-level defaults and method-level overrides

Timer Service

- Added in EJB 2.1 Specification
- Persistent
  - Timers survive server shutdown/restart
- Transactional
  - Timer operations (create/cancel/timeout) are first-class transactional units of work
- Intended to model long-lived business processes
- Timers created via **`javax.ejb.TimerService`** API
  - One-time expiration or at fixed recurring intervals

Timer Example

```
@Stateless public class AccountBean implements Account {

    @Resource TimerService timerSvc;
    @PersistenceContext EntityManager accountDB;

    public Integer createNewAccount(Details details) {
        Integer acctNum = ...;
        // Create new account
        ...

        // Initial deposit must be made within 10 days
        timerSvc.createTimer(FUNDING_TIMELIMIT,
acctNum);     }

    @Timeout void verifyFunding(Timer t) {
        // Verify that initial account deposit cleared
        ...
    }
}
```

Use Case: Generate Monthly Bank Statements

- A bank needs to generate checking/savings account statements the 1st of each month and email them to account holders
- Like a UNIX “cron job” where the work to be performed at timeout is a good fit for an EJB component
    - Transactions
    - Database access
    - JavaMail™ API

Monthly Bank Statements Example Using EJB 3.0 Technology

```
public class InitEvents implements ServletContextListener
{

    @EJB Accounts accounts;

    // Called whenever web application initializes
    public void contextInitialized(ServletContextEvent e)
    {
        accounts.createTimer();
    }

}
```

Monthly Bank Statements Example Using EJB 3.0 Technology

```
@Stateless public class AccountBean implements Accounts {
    @Resource TimerService timerSvc;
    @Resource javax.mail.Session mailSession;
    @PersistenceContext EntityManager accountDB;

    public void createTimer() {
        if ( timerSvc.getTimers().size() == 0 ) {
            long timeUntilFirstOfNextMonth = ...;


timerSvc.createTimer(timeUntilFirstOfMonth,...);
        }
    }

    @Timeout void timerExpired(Timer t) {
        // Generate and send monthly bank accounts
        ...
    }
}
```

Lessons Learned

- Difficult to configure calendar-based events using only relative time units
- How to register the timer in the first place?
- Typical container initialization events (Web Application contextInitialized(), Servlet.init, etc.) are not a great fit
  - They happen every time application initializes and in every server instance
    - Burden is on developer to check if timer already exists
    - No way to guarantee that only one is created per cluster

Calendar-Based Timer Scheduling

- Should be able to express timer expiration based on a calendar instead of relative to creation time
  - "The second day of every month at noon"
  - "Every Wednesday at five a.m."
  - "Every half-hour on Saturdays and Sundays"
- Exact syntax is TBD
  - Probably cron-"like"
  - Cron syntax well-defined/widespread but difficult to read
    - "0   12   *   2   2"   == "Every Tuesday in February"

Automatic Timer Creation

- Create a timer as a result of deploying an application
  - Useful for registering a one-time or recurring application-specific action independent of a business method invocation
    - e.g., “Generate bank statements the 1st of every month”
    - For each timer expiration, callback happens in one server instance, not every server instance in cluster
- Approaches
  - Specify via meta-data (annotation or .xml)
  - Define callback that happens once per application deployment and call **`TimerService.createTimer`** within it

Automatic Timer Creation Example

```
// Create a timer for the 1st day of each month at noon
@EJBTimer("0 12 1 * *", "statementTimer")

@Stateless public class AccountBean {

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

    @Timeout void sendMonthlyStatements(Timer t) {

        // Calculate monthly bank statements and
        // send them out via email
        ...

    }
}
```

Asynchronous Support in EJB 3.0 Specification

- Java APIs for XML Web Services (JAX-WS)/Stateless WebService Endpoint
  - Asynchronous request/response, @OneWay
  - Good for web services applications, but too cumbersome to use for simple intra-application asynchrony
- Java Message Service (JMS)/Message-Driven Beans
  - Good for loose coupling/guaranteed delivery semantics
    - For intra-application asynchrony, messaging API still too complex compared to procedural invocation
- Timer Service
  - Create single-action timer with "immediate" expiration time to convert synchronous operation to asynchronous
    - Not the intended usage

Simple Asynchronous Operations

- Use metadata to mark a Local/Remote business method as asynchronous
- Container returns control to client before executing business method
- No separate API to learn

```
@Asynchronous public void doSomething(Details d) {
    // ...
}
```

Async Operation + Task Status Using Stateful Session Bean

```
@Stateful @BeanManagedConcurrency
public class AsyncTaskBean implements AsyncTask {

   private boolean taskComplete = false;

   @Asynchronous public void doSomething(Details d) {
      // perform work
      ...

      taskComplete = true;
   }

   public boolean isTaskComplete() {
      return taskComplete;
   }
}
```

Stateful Web Service Endpoints

- EJB 3.0 specification supports stateless Web Service endpoints via Stateless Session Beans
  - Based on JAX-WS API/Java Architecture for XML Binding (JAXB)
- Stateful interaction useful to web service clients as well
- Allow Stateful Session Beans to be exposed as web service endpoints
  - Details of client programming model/stateful identity propagation defined by JAX-WS Specification

Stateful EJB Web Service Endpoint

```
@WebService
@Stateful
public class CartBean {

    private Collection<Item> items;

    public void addItem(Item i ) { ... }
    public void removeItem(Item i) { ... }

    @Remove public void checkout(...) { ... }
    @Remove public void cancel() { ... }

}
```

Agenda

Overview

Ease of Use Enhancements

New Features

Summary

Q&A

Summary: Enterprise JavaBeans 3.1 Technology

- Part of Java EE platform 6
- Further simplify EJB component development
  - Optional Local Business Interfaces
  - EJB components in the Web Tier
- Add new features
  - Singletons
  - Concurrency Options
  - Timer Service Enhancements
  - Simple Asynchrony
  - Stateful Web Service Endpoints

Related Sessions/BOFs

- Java Persistence 2.0
  - Linda DeMichiel
  - Wednesday, 10:55AM–11:55AM
  - TS-4945
- Java EE Platform: Meet the Experts BOF
  - Wednesday night, 7:55PM–9:45PM
  - BOF-4641, BOF-4642

Q&A

Enterprise JavaBeans™ 3.1 Technology

Kenneth Saks

Senior Staff Engineer

Sun Microsystems

TS-4247