Java Persistence API

RESTful Web Services
JAX-RS

Technical Introduction

Matthew Perrins
Executive IT Specialist
Martin Gale
Consulting IT Specialist

Overview of Talk

- Java and Persistence
- Technology Overview
- Database Schema and Example Code
- JPA Deep Dive
- JPA Development and Test Considerations
- General Comments

- What is REST
- JAX-RS Introduction
- JAX-RS Deep Dive
- Conclusion
- Q & A

Modern JEE Architecture

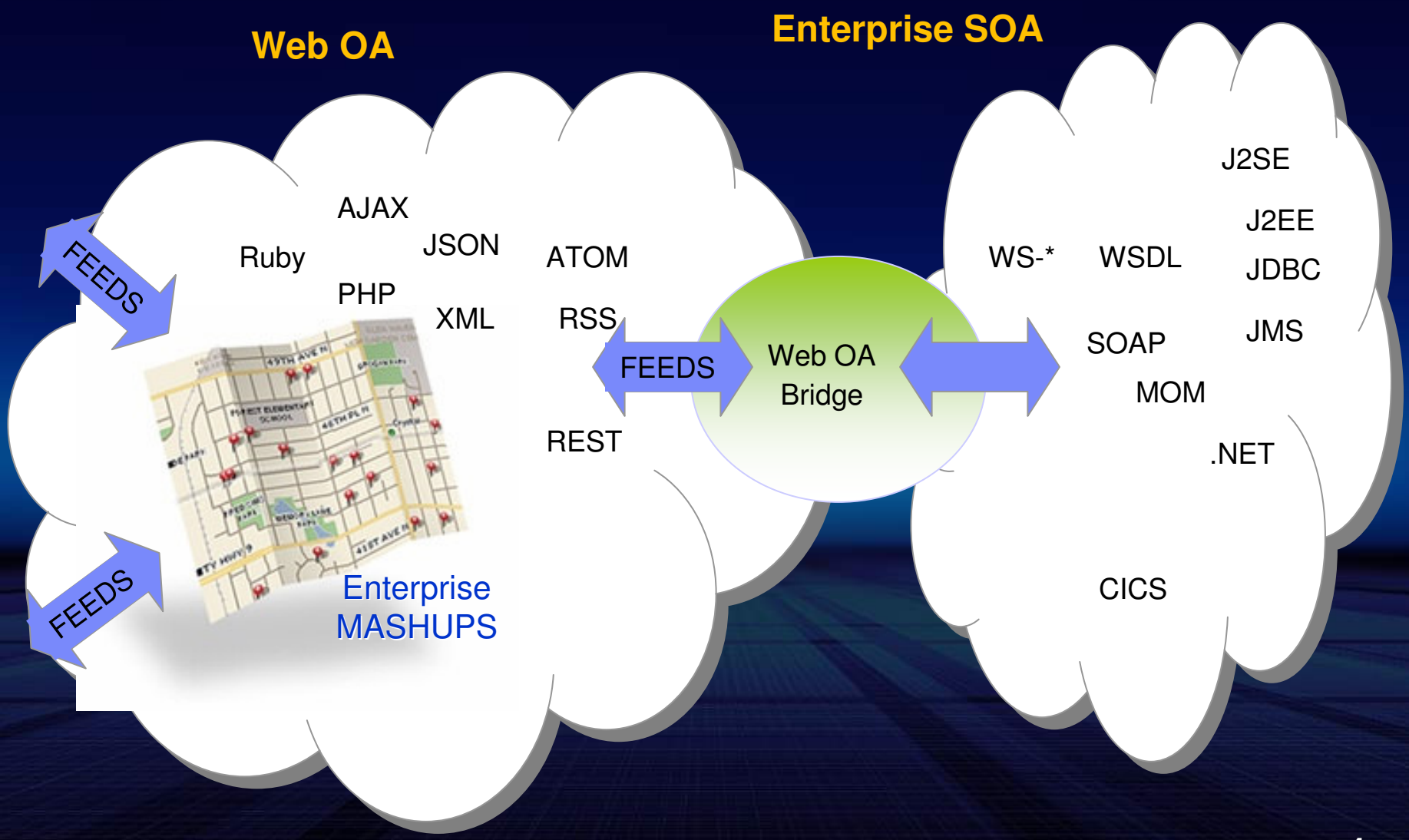
Specifications

HTML->Dojo->XHR->JSON->JAX-RS->JPA->JDBC

Runtime

Firefox 3.0->Dojo Toolkit->Apache JAX-RS->WAS 7.0->DB2

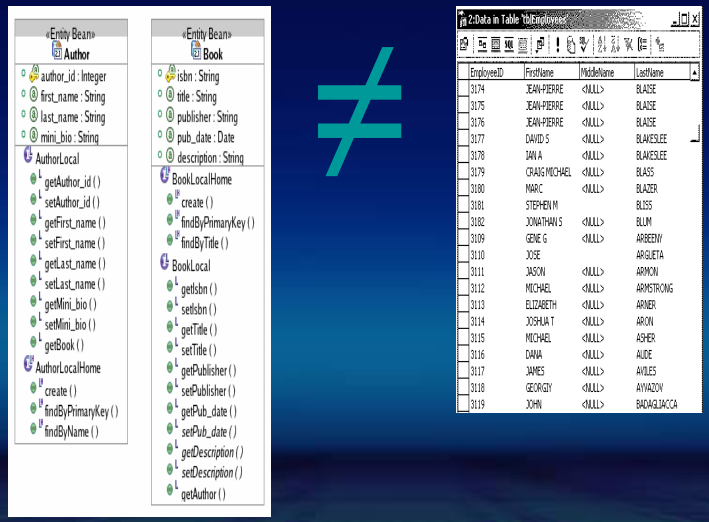
Why ? - Exposing Data into the WOA



Persistence?

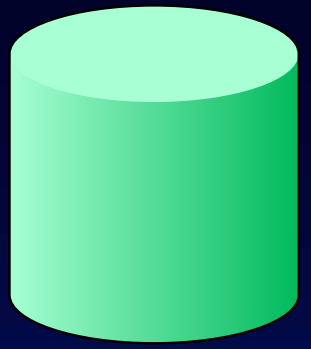
- "The capability to store data structures in non-volatile storage" — Wikipedia
- Why is this non-trivial in an Enterprise Java Application?

Java Objects ≠ **Relational Entities**



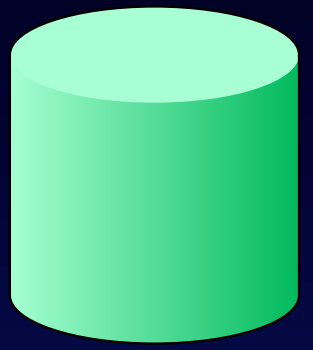
- The Solution?
  - It depends on the application, but often involves domain modeling outside the database to map relational database constructs to object-oriented constructs or **O/R Mapping**

Java and Persistence



- How do we persist data in Java?
  - Serialization
  - Database
    - Relational DB
    - OO DB
- Java EE focus
  - Relational Database is dominant Data Source
  - XML used for integration and services scenarios
  - Other legacy datastores
    - Mainframe applications (CICS, IMS, VSAM, etc…)
- JDBC API
  - Major API for accessing Relational databases
  - Allows you to pass SQL to database
  - Represents Data in a ResultSet
  - Efficient, but specific to the database

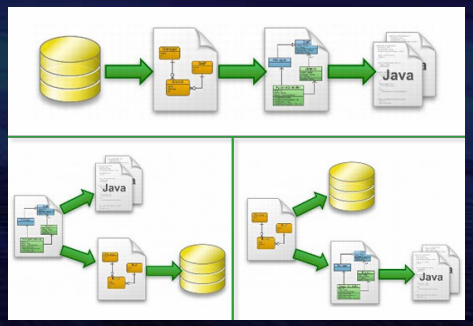
Data Persistence Evolution



- Plain JDBC
  - Usually requires custom coding and leads to home grown frameworks.
- EJB 2.x CMP Entity Beans
  - CMP is *perceived* as failure in the market place.
    - Bad press (serverside.com)
    - Mapping done completely different from vendor to vendor, defeats portability gain.
  - Programming model is heavy weight.
    - Too many interfaces.
    - Objects always connected, requires extra Data Transfer Object Layer in almost every case.
- JPA (Java Persistence API)
  - EJB 3 Specification selected in favor of JPA (over CMP model)
    - JPA moves toward patterns from TopLink, Hibernate, and JDO

ORM (Object Relational Mapping)

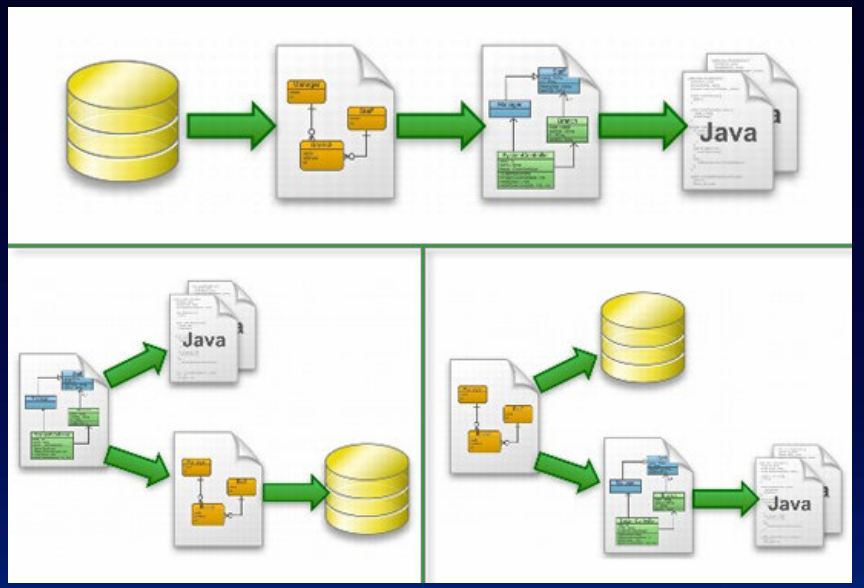
- Framework/tooling around providing automatic persistence
- Objects are mapped to Tables using Metadata
- Metadata is used to transform data from one representation to another.
- ORM necessary for QOS in persistence
  - Caching, identity management, operational environment, querying system, transparency, isolation, etc…



Overview of Talk

- Java and Persistence
- Technology Overview
- Database Schema and Example Code
- JPA Deep Dive
- JPA Development and Test Considerations
- General Comments
- Q & A

Technology Overview



- Light ORM
  - JDBC RowSet API
  - SQLJ
  - Apache iBatis 2.0
  - pureQuery API's
- Full ORM
  - **Java Persistence API (JPA)**
  - Java Data Objects (JDO) 2.0
  - Hibernate 3.X
- Persistence for IBM SOA products
  - pureXML
  - WBI JDBC and EIS Adapters

Java Persistence API (JPA)

- Standard persistence technology for Java Enterprise Edition 5
  - A form of O/R mapping (called container managed persistence in Java EE 1.3/4)
  - Designed for highly distributed applications
  - Lifecycle manageable by application server to increase quality of service
- EJB specification made very significant changes:
  - Standardizes O/R mapping metadata (not the case for EJB 2.x)
  - Java SE 5.0 Annotations can be used instead of XML deployment descriptor
  - No deploy code implementing abstract classes— Entities are POJOs
  - Application Server is not required
    - The Java Persistence API is available outside of Containers
    - Unit testing greatly simplified
  - Removal of checked exceptions (for instance, remote exceptions)
  - No bean (or business) interface required

Standards –JavaSE & JavaEE

- Java Community Process
  - JSR-220
  - Founded by
  - Part of Java Enterprise Edition 5.0 and EJB 3.0
  - Spec leads: Linda DeMichiel and Mike Keith
- The specification has three documents:
  - EJB Simplification API document
  - EJB Core Contracts
  - **JPA 1.0**
- Updated specification in the works…
  - JSR-317 – **JPA 2.0**
  - Member of Java Enterprise Edition 6.0
  - Spec lead: Linda DeMichiel

Dependencies and Data Store Support

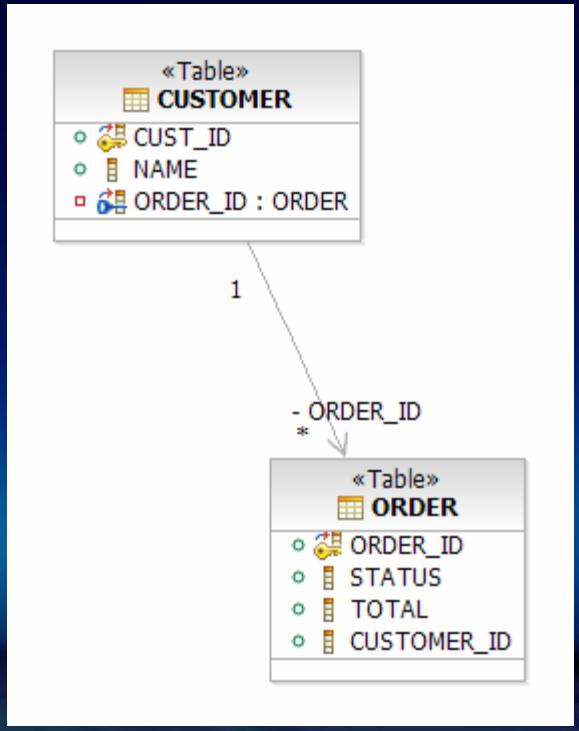
- Java SE 5.0 or Java EE 5.0
  - The persistence API is supported outside the application server
  - Dependencies are likely to depend on the provider or vendor
- Data store support – rich support
  - Entities can be written to interact with any underlying data source:
    - Databases (native support)
    - XML Repositories (custom mapping)
    - LDAP (custom mapping)
    - Others…

Overview of Talk

- Java and Persistence
- Technology Overview
- Database Schema and Example Code
- JPA Deep Dive
- JPA Development and Test Considerations
- General Comments
- Q & A

Database Schema & Code Examples

- We'll be using a small subset of a sample database schema:



- The code examples implement submitOrder():
  - Retrieves the Order related to Customer
  - Updates the Order's status to "SUBMITTED"
  - Removes the Customer-Order Relationship

Overview of Talk

- Java and Persistence
- Technology Overview
- Database Schema and Example Code
- JPA Deep Dive
- JPA Development and Test Considerations
- General Comments
- Q & A

JPA Persistence Feature Support

- Types –Entities
  - Entities are Plain Old Java Objects (POJOs) with Annotations…
  -

```
@Entity
@Table(name="CUSTOMER")
public class Customer implements Serializable {
```

  - Or, they can be supported by mapping files

Java File: **Customer.java**
public class Customer implements Serializable {…}

Mapping file: **customer.xml**

```
<entity class="Customer" >
   <table name="CUSTOMER"  />

…
</entity>
```

JPA Persistence Features (continued)

- Key Attributes –annotated fields and key classes
  - @Id annotation can be used to signify a single key, or a key class can be used
  - Composite Keys
  - Fully supports key generation
    - Identity Columns
    - Sequence
    - Table Generator
    - Auto (persistence provider determines)

```
@Id @GeneratedValue(strategy=GenerationType.IDENTITY)
@Column(name="ORDER_ID")
public Integer getOrderId() {
   return id;
}
```

  - Or, XML mapping can be used…

```
<attributes>
   <id name="id">
        <column name="ORDER_ID"/>
        <generated-value strategy=IDENTITY/>
   </id>
 </attributes>
```

JPA Persistence Feature Support

- Attributes –Java properties
  - Annotations can be specified per Java property (or field) to bind to database columns:

```
@Column(name="DESC")
public String getDescription() {
                return description;
}
```

  - Or, attributes can be specified via XML mapping files

```
<attributes>
   <basic name="description">
      <column name="DESC" />
   </basic>
</attributes>
```

JPA Persistence Features (continued)

- Derived Attributes –supported
  - Fields that are marked as @Transient
  - The getter for the field can then contain the logic to calculate/enforce it
  - Optional Entity Listeners and callbacks (similar to EJB 2.x callbacks) exist to maintain derived attributes throughout the lifecycle of persistence

```
@Transient
public long getRatio() {
                ratio = checkingBalance / savingsBalance;
                return ratio;
}
```

  - Or, via XML mapping file…

```
<attributes>
   <transient name="ratio"/>
</attributes>
```

JPA Persistence Features (continued)

- Relationships
  - Attributes can be other entities or java.util.Collections of other entities
  - Relationships can be traversed, all relationships are unidirectional by default
  - N-to-N, 1-to-N, and 1-to-1.

```
@OneToOne(cascade=CascadeType.ALL , fetch=FetchType.EAGER )
@JoinColumn(name="OPEN_ORDER_ID", referencedColumnName="ORDER_ID")
public CustomerOrder getCurrentOrder() {
                return currentOrder;
}
Or

<attribute name="currentOrder">
    <one-to-one fetch="EAGER" >
        <cascade><cascade-all/></cascade>
    </one-to-one>
    <join-column name="OPEN_ORDER_ID" referencedColumnName="ORDER_ID" />
</attribute>
```

```
@OneToOne(mappedBy="currentOrder", cascade=CascadeType.ALL ,
        fetch=FetchType.EAGER)
                public Customer getCustomer {
                                ...
```

  - Can be made bi-directional

JPA Persistence Features (continued)

- Entity Listeners and Life Cycle callbacks (Optional) annotations or XML
  - @PrePersist
  - @PostPersist
  - @PreRemove
  - @PostRemove
  - @PreUpdate
  - @PostUpdate
  - @PostLoad
- On Entity class or external class

```
@Entity
@EntityListeners(com.acme.AlertMonitor.class)

public class Account {
        Long accountId;
        Integer balance;
        boolean preferred;
        @Id
        public Long getAccountId() { ... }
        public Integer getBalance() { ... }
...
        @Transient // because status depends upon non-persistent context
        public boolean isPreferred() { ... }

        public void deposit(Integer amount) { ... }
        public Integer withdraw(Integer amount) throws NSFException {... }

        @PrePersist
        protected void validateCreate() {
                if (getBalance() < MIN_REQUIRED_BALANCE)
                throw new AccountException("Insufficient balance to open an
                account");
        }

        @PostLoad
        protected void adjustPreferredStatus() {
                preferred =
                (getBalance() >= AccountManager.getPreferredStatusLevel());
        }
}
```

```
public class AlertMonitor {
@PostPersist
public void newAccountAlert(Account acct) {
        Alerts.sendMarketingInfo
        (acct.getAccountId(),
        acct.getBalance());
}
```

JPA Persistence Metadata

- Java SE 5.0 Annotations
- persistence.xml
  - Defines persistence unit and the datasource
- orm.xml
  - Defines additional object/relational mapping information

```
<persistence-unit name="db2pu">
        <jta-data-source>jdbc/DB2DS</jta-data-source>
        <mapping-file>META-INF/customerORM.xml</mapping-file>
        <properties>
                <property name="openjpa.Log" value="DefaultLevel=TRACE"/>
        </properties>
</persistence-unit>
```

JPA Persistence Programming Model

- Initializing the framework
  - Entity Manager injected via annotation:
    - Only for Container Managed Entity Managers in Java EE Containers.
    - For Application Managed Entity Managers, use EntityManagerFactory create method :

```
@Stateless
public class OrderEntryFacadeBean implements OrderEntryFacade {
        @PersistenceContext (unitName="db2pu")
         private EntityManager em;
        ...
}
```

- Making the Connection –Entity Manager
  - Entity manager transparently handles connections to the data store

JPA Programming Model (continued)

Transaction Demarcation – automatic (when used within CMT of Session Bean/MDB) or explicit…

- Within EJB Container, Session Bean/MDB using CMT
  - Entities themselves no longer have CMT, but work within CMT

```
@Stateless
public class OrderProcessorBean implements OrderProcessor {
@PersistenceContext (unitName="db2pu")
private EntityManager em;

@TransactionAttribute (REQUIRED)
   public void submit(int customerId)
   throws CustomerDoesNotExist, OrderNotOpen
   {
```

- Java Transaction API (JTA) can be used directly:

```
UserTransaction tx = //INJECT, LOOKUP, or EJBCONTEXT
tx.begin();
//code
tx.commit();
```

- EntityTransaction for local transaction support:
  - For example, Java SE applications

```
DataObject param = mediator.getParameterDataObject();
EntityTransaction tx = em.getTransaction();
tx.begin();
//code
tx.commit();
```

JPA Programming Model (continued)

- Create –like POJO
  - Creating an entity bean is as easy as creating a plain Java object:

```
CustomerOrder newOrder = new CustomerOrder();
newOrder.setStatus("OPEN");
newOrder.setTotal(new Integer(0));
newOrder.setCustomerId(customerId);
em.persist(newOrder);
```

JPA Programming Model (continued)

- Retrieve –various
  - You can find by primary key with the entity manager:

```
Customer customer = (Customer) em.find(Customer.class, customerId);
```

  - EJB / JP-QL query string:

```
Query q = em.createQuery("SELECT c FROM Customer c WHERE c.name LIKE :custName");
q.setParameter("custName", name);
q.setMaxResults(10);
List result = q.getResultList();
```

  - Native SQL query string:

```
Query q = em.createNativeQuery (
        "SELECT o.id, o.quantity, o.item, i.id, i.name, i.description "+
        "FROM Order o, Item i " +
        "WHERE (o.quantity > 25) AND (o.item = i.id)",
        "OrderItemResults");

        @SqlResultSetMapping(name="OrderItemResults",entities={
        @EntityResult(entityClass=com.acme.Order.class),
        @EntityResult(entityClass=com.acme.Item.class)}
)
```

JPA Programming Model (continued)

- Named Queries
  (Both JP-QL or Native supported)

```
@Entity
@Table(name="ORDER")
@NamedQuery(name="getAllOrders", query="SELECT o FROM CustomerOrder o " +
                                       "WHERE o.customerId = :customerId")
public class CustomerOrder implements Serializable {
```

  - Or …

```
<named-query name="getAllOrders">
    <query>SELECT o FROM CustomerOrder o WHERE o.customerId = :customerId</query>
</named-query>
```

  - Can be invoked like…

```
Query allOrders = em.createNamedQuery("getAllOrders");
allOrders.setParameter("customerId", customerId);
Collection <CustomerOrder>  results = allOrders.getResultList();
return results;
```

JPA Programming Model (continued)

- Update –automatic or by query
  - When a property of an entity bean is set, the entity manager automatically intercepts the changes
  - If an object was loaded as part of the context, no save is needed – the transaction commit will automatically persist any changes.
  - Other methods
    - em.persist(…);
    - em.merge(…); //For detached instances
  - Update Query

```
String ejbqlUpdate = "update Customer set name = :newName where name = :oldName"
int updatedEntities = entityManager.createQuery( ejbqlUpdate )
                        .setParameter( "newName", newName )
                        .setParameter( "oldName", oldName )
                        .executeUpdate();
```

- Delete –Explicit remove
  - JP–QL Delete, similar to update

```
em.remove(myEntityBean);
```

JPA Programming Model (continued)

- Error Handling – unchecked exceptions
  - EJB 3 philosophy is not to use checked exceptions
  - Example of runtime exception that can be caught when an entity reference is not found

```
LineItem existingLineItem = null;
try
{
        LineItemId lineItemId = new LineItemId();
        lineItemId.setOrderId(existingOrder.getOrderId());
        lineItemId.setProductId(productId);
        existingLineItem = (LineItem)em.getReference(LineItem.class, lineItemId);
} catch (EntityNotFoundException e) {
        throw new NoLineItemException();
}
```

JPA Customer Class Code Example

```
@Entity
@Table(name="CUSTOMER")
public class Customer implements Serializable {
        private int id;
        private String name;
        private Order openOrder;

        @Id
        @Column(name="CUST_ID")
        public int getId() { return id; }
        public void setId(int id) { this.id = id; }

        @Column(name="NAME")
        public String getName() { return name; }
        public void setName(String name) { this.name = name; }

        @OneToOne(cascade=CascadeType.ALL , fetch=FetchType.EAGER )
        @JoinColumn(name="OPEN_ORDER_ID", referencedColumnName="ORDER_ID")
        public Order getOpenOrder() { return openOrder; }
        public void setOpenOrder(Order openOrder) { this.openOrder = openOrder; }
}
```

JPA Order Class Code Example

```
@Entity
@Table(name="ORDER")
public class Order implements Serializable {
        private int id;
        private String status;
        private Customer customer;

        @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
        @Column(name="ORDER_ID")
        public Integer getId() { return id; }
        public void setId(int id) { this.id = id; }

        @Column(name="STATUS")
        public String getStatus() { return status; }
        public void setStatus(String status) { this.status = status; }

        @OneToOne(fetch=FetchType.EAGER, optional=false)
        @JoinColumn(name="CUSTOMER_ID", referencedColumnName="CUST_ID",
            insertable=false, updatable=false, nullable=false, unique=true)
        public Customer getCustomer() { return customer; }
        public void setCustomer(Customer customer) { this.customer = customer; }
}
```

JPA Persistence Customer/Order Code Example

```
@PersistenceContext (unitName="db2pu")
private EntityManager em;

 @TransactionAttribute (TransactionAttributeType.REQUIRED)
 public void submit(int customerId) throws CustomerDoesNotExist, OrderNotOpen
 {
        // Get the Customer
        Customer customer = (Customer) em.find(Customer.class, new Integer(customerId));
        if (customer==null) {
            throw new CustomerDoesNotExist(customerId);
        }
        // Get Customer's related Order
        Order openOrder = customer.getOpenOrder();
        if (openOrder == null) {
            throw new OrderNotOpen(customerId);
        }
        // Remove Order relationship from Customer and change Order status to submitted
        customer.setOpenOrder(null);
        openOrder.setStatus("SUBMITTED");
}
```

Overview of Talk

- Java and Persistence
- Technology Overview
- Database Schema and Example Code
- JPA Deep Dive
- JPA Development and Test Considerations
- General Comments
- Q & A

JPA Notepad Development Process

- Build Considerations
  - Entity Beans are packaged in a JAR file in 3.0
  - The build environment must be able to compile according to this packaging (in other words support the JAR spec)
  - ANT and/or Maven is often used for building
- Unit Testing
  - Since JPA 1.0 Entities can run in Java SE 5 with a “light weight” entity manager, just the JVM is needed for unit testing and the implementation library (the provider of the implementation may require additional dependencies)
  - Testing inside the container is vendor dependent (setup, configuration, etc).
- Deployment Process
  - Vendor/Provider dependent

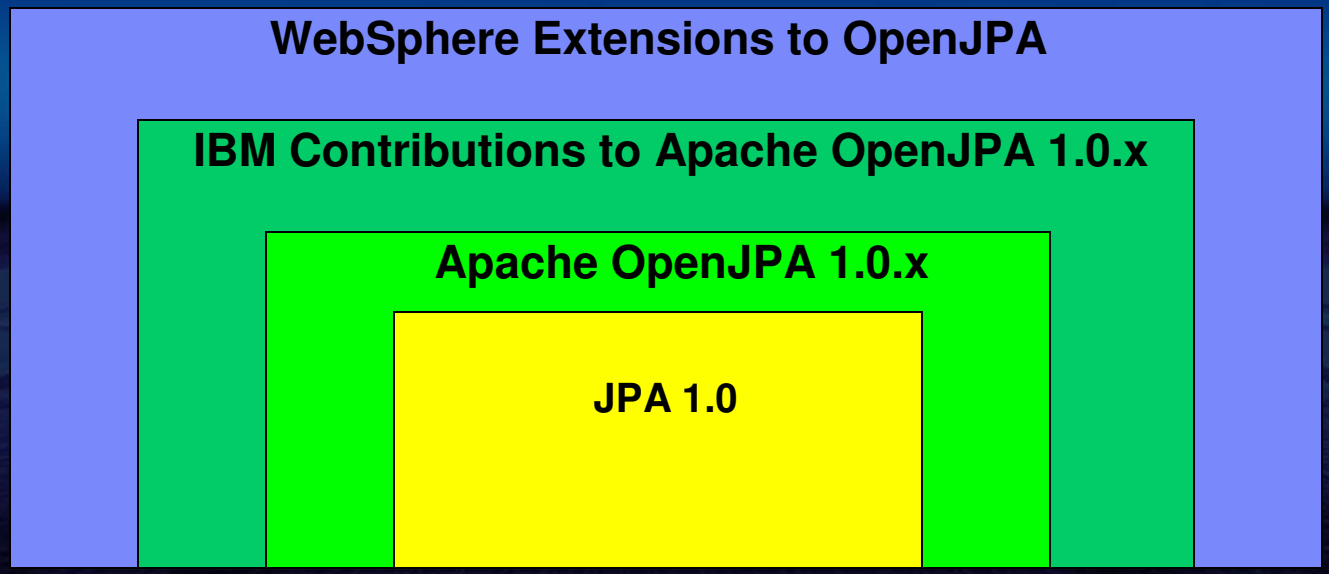
Java SE vs Java EE Test Differences

- Persistence Unit
  - Transaction Type
    - JTA (Java EE default)
    - RESOURCE_LOCAL (Java SE default)
  - DataSource
    - jta-data-source (requires JNDI)
    - non-jta-data-source (requires JNDI)
    - provider-specific properties (ie. openjpa.ConnectionURL)
  - Provider
    - specify in persistence unit (clearest)
    - default provider configuration in App Server environment
    - rely on classpath (providers export themselves via a Services entry)
  - Entity Listings
    - most portable – list all classes (maintenance headache)
    - most providers can detect Entity classes

Architecture of WebSphere's JPA Solution

WebSphere's JPA Solution

- Spec Compliant
- Feature Rich
- Extensible
- Competitive with Hibernate and EclipseLink



General Comments

- WebSphere Application Server v7
  - http://www-01.ibm.com/software/webservers/appserv/was/
- WebSphere Application Server v6.1 Feature Pack for EJB 3.0
  - http://www-1.ibm.com/support/docview.wss?rs=177&uid=swg21287579
- Apache OpenJPA code
  - IBM partnering with Apache OpenJPA Community on enhancing and supporting OpenJPA
    - http://openjpa.apache.org/
- White papers
  - http://www.ibm.com/developerworks/websphere/techjournal/0712_barcia/0712_barcia.html
  - http://www.ibm.com/developerworks/websphere/techjournal/0612_barcia/0612_barcia.html

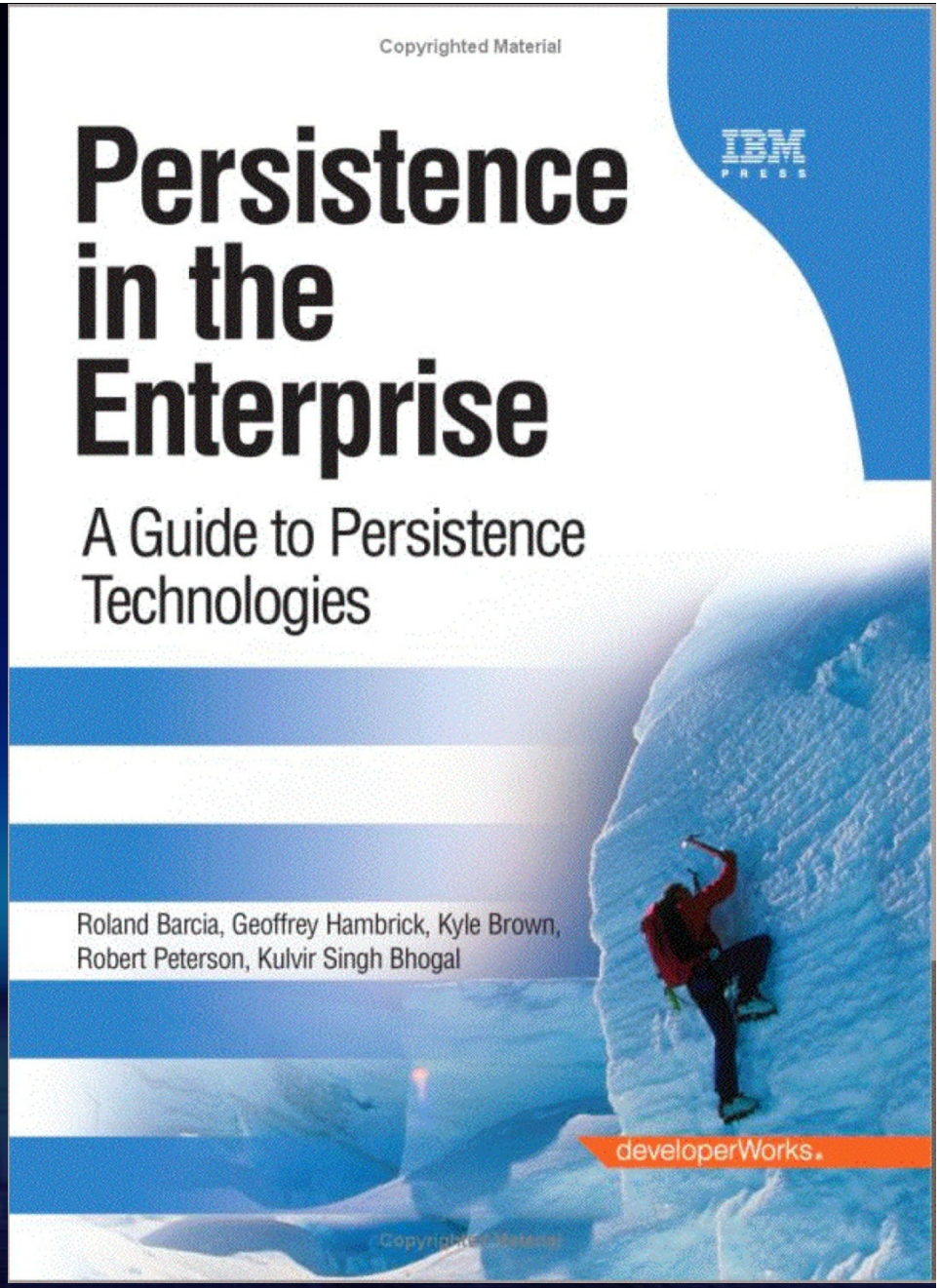
General Comments

- Tooling
  - RAD 7.5 works with WebSphere Application Server v7 and WebSphere v6.1 + EJB 3 FP
  - Eclipse Dali plugin
    - http://www.eclipse.org/webtools/dali/main.php
    - Nice plugin aide for development of JPA Entities and associated xml configuration files
    - Part of WebTools 2.0 (requires Eclipse 3.3)

Book from my team

- Persistence in the Enterprise.
  - Roland Barcia, Geoffrey Hambrick, Kyle Brown, Robert Peterson, and Kulvir Bhogal

- Taking an end-to-end application architecture view of persistence
- Understanding business drivers, IT requirements, and implementation issues
- Driving your persistence architecture via functional, nonfunctional, and domain requirements
- Modeling persistence domains
- Mapping domain models to relational databases
- Building a yardstick for comparing persistence frameworks and APIs
- Selecting the right persistence technologies for your applications
- Comparing JDBC™, Apache iBATIS, Hibernate Core, Apache OpenJPA, and pureQuery



JSR 311 RESTful Web Services

Also know as JAX-RS

Introduction

Flamewar SOAP vs REST

- “Web services are too complex”
- “How do you deal with things like transactions, reliability, etc?”
- “Web services are reinventing the Web through HTTP POST”
- “Foolish to model everything as GET,POST, PUT and DELETE operations”
- “Losing benefits of the Web”

Why SOAP can be hard - Example

- Issue comes from example 1 in the SOAP 1.1 specification:

```
• POST /StockQuote HTTP/1.1
SOAPAction: "Some-URI"
<SOAP-ENV:Envelope xmlns:SOAP-ENV="...">
        <SOAP-ENV:Body>
                <m:GetLastTradePrice xmlns:m="Some-URI">
                        <symbol>DIS</symbol>
                </m:GetLastTradePrice>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- **Overview**
  - Goes against the Web architecture: simple
  - information retrieval is done with GET
  - http://stockquoteserver.example/query?symbol=DIS
  - One URI for every message
  - Not book markable
  - Potentially performance issues
  - Not cacheable
  - Potentially big payload

REpresentational State Transfer

- Architectural style for distributed hypermedia
- systems, introduced by Roy Fielding (2000)
  - Client-Server
  - Stateless communication
  - Cacheability
  - Uniform interface for resources
  - Layered System
  - Code-on-demand (optional)

- **REST Concepts**
- Resources (nouns)
- Identified by a URI, For example:
  - http://www.parts-depot.com/parts
- Connectedness
- Uniform interface (verbs)
- Small fixed set:
  - Create, Read, Update, Delete
- State Representations
  - data and state transferred between client and server
  - XML, JSON, Atom, XHTML, ...

URI Templates – Rest Entities

- Orders
  - **/rest/orders – list of all the orders**
  - **/rest/orders/{id} - 1 order with the given order id**
  - **/rest/orders/{id}/lineitems**
- URI Templates are URIs with variables within the URI syntax.
- Orders can be returned in any standard web format


public class Artist extends HttpServlet {
public enum SupportedOutputFormat {XML, JSON};
protected void doGet(HttpServletRequest request,
throws ServletException, IOException {
String accept = request.getHeader("accept").
String acceptableTypes[] = accept.split(",");
SupportedOutputFormat outputType = null;
for (String acceptableType: acceptableTypes)
if (acceptableType.contains("*/*") ||
acceptableType.contains("applica
outputType=SupportedOutputForma
break;
} else if (acceptableType.conta
outputType=SupportedOutpu
break;
}
}
if (outputType==null)
response.sendError(
String path = request
String pathSegment
String artist = pa
if (pathSegments
response.
else if (path
if (out
els
} e
void write
priv
private
private voi
}
REJECTED

REST in Java…the JAX-RS way

- Standardized in the JCP
  - JSR 311
- Will be included in Java EE 6 - Group started in April 2007
  - Marc Hadley, Paul Sandoz
- EG members
  - Alcatel-Lucent, BEA, Day Software, Fujitsu, IBM, innoQ, Nortel, Red Hat
- Experts in Atom, AtomPub, WebDAV, HTTP, REST, Restlet

JAX-RS Introduction

- Set of Java APIs for development of web services built according to the REST principals
- Complete Today 3rd March 2009

JSRs: Java Specification Requests

JSR 311: JAX-RS: The Java™ API for RESTful Web Services

This JSR will develop an API for providing support for RESTful(Representational State Transfer) Web Services in the Java Platform.

Status: Final

Stage		Start	Finish
Maintenance Draft Review	Download page	27 Jan, 2009	03 Mar, 2009
Final Release	Download page	10 Oct, 2008	
Final Approval Ballot	View results	09 Sep, 2008	22 Sep, 2008
Proposed Final Draft	Download page	15 Aug, 2008	
Public Review Ballot	View results	27 May, 2008	02 Jun, 2008
Public Review	Download page	02 May, 2008	02 Jun, 2008
Early Draft Review	Download page	24 Oct, 2007	23 Nov, 2007
Expert Group Formation		27 Feb, 2007	15 Aug, 2007
JSR Review Ballot	View results	13 Feb, 2007	26 Feb, 2007

JCP version in use: 2.6
Java Specification Participation Agreement version in use: 2.0
Please direct comments on this JSR to: jsr-311-comments@jcp.org

Specification Lead

Marc Hadley Sun Microsystems, Inc.
Paul Sandoz Sun Microsystems, Inc.

Expert Group

Algermissen, Jan
De Hora, Bill
Harby, John
Hensley, David
Koops, Stephan
McDonough, Ryan J.
Prasanna, Dhanji R.
Schulz-Hofen, Jan
BEA Systems
Fujitsu Limited
He, Hao Dr.
IBM
Louvel, Jerome
NCsoft Corporation
Red Hat Middleware LLC
Sun Microsystems, Inc.
Day Software, Inc.
Grabovac, Nickolas
Heaton, Ryan
innoQ Deutschland GmbH
Mark Hansen
Nortel
Reschke, Julian

Install Dependant JARs into WEB-INF/lib

- The IBM implementation can work on Geronimo and WebSphere
  - Add the following JARs to you WEB-INF/lib directory

```
abdera-0.4.0-incubating.jar
abdera-core-0.4.0-incubating.jar
axiom-api-1.2.5.jar
axiom-impl-1.2.5.jar
commons-codec-1.3.jar
commons-httpclient-3.1.jar
commons-logging-1.1.1.jar
geronimo-annotation_1.0_spec-1.1.1.jar
geronimo-servlet_3.0_spec-1.0-EA-20080711.180136-1.jar
jaxen-1.1.1.jar
jettison-1.0.1.jar
jetty-6.1.9.jar
jsr311-api-0.11.jar
xmlbeans-2.3.0.jar
```

Configure JAX-RS Servlet

- Add the following Servlet definition to your WEB.XML Web Project Descriptor.
  - You can define your root URL-Pattern
  - **rest** or **resources** are common patterns used
  - Or set the context root of your web project to **rest**

```
<servlet>
          <servlet-name>JAX-RS Servlet</servlet-name>
          <servlet-class>com.ibm.ws.jaxrs.web.RESTServlet</servlet-class>
          <init-param>
                    <param-name>javax.ws.rs.Application</param-name>
                    <param-value>com.ibm.prism.rest.PrismApplication</param-value>
          </init-param>
</servlet>

<servlet-mapping>
          <servlet-name>JAX-RS Servlet</servlet-name>
          <url-pattern>/*</url-pattern>
</servlet-mapping>
```

Create Application Class

- To define the entry points for the Servlet you need to define what is called an Application Class, this loads the Rest classes uses within the WAR.
- Maps to the `<param-name>javax.ws.rs.Application</param-name>` on the servlet

`com.ibm.prism.rest.PrismApplication`

```
package com.ibm.prism.rest;
import java.util.HashSet;
import java.util.Set;

import javax.ws.rs.core.Application;
public class PrismApplication extends Application {
    public Set<Class<?>> getClasses() {
        Set<Class<?>> classes = new HashSet<Class<?>>();
        classes.add(ObjectResource.class);
        classes.add(SolutionResource.class);
        return classes;
    }
}
```

Resource Methods

- Method of a resource class annotated with @HttpMethod
- Standard implementation for @GET, @POST,@PUT, @DELETE, @HEAD

- Extendable
  - i.e. WebDav methods Move, Copy, Search
  - Parameters can be annotated
  - Examples: @PathParam, @CookieParam, @HeaderParam,..

- Return Types
  - Resource methods MAY return **void,**
- **Response or another Java type**
  - Conversion from Java type to entity body by an entity provider
- Exceptions
  - **WebApplicationException**

Resource Example

- Using Annotations you can easily define the methods that handle the common REST actions of GET,POST, DELETE, PUT
- @path defines the resource entity reference relative to context root
  - e.g. `/rest/objects`

`com.ibm.prism.rest.ObjectResource`

```
package com.ibm.prism.rest;

@Path(value="/objects")
@Produces("application/json")
public class ObjectResource {

    @Produces(value={"application/json"})
    @GET
    public Response read()
    @DELETE
    public Response delete()
    @PUT
    public Response update()
    @POST
    public Response create()
}
```

Representations

- •Supported requests and response media types can be declared by
- – @ConsumeMime("application/xml")
- – @ProduceMime("application/json")
- Annotations may be applied to
  - Resource classes
  - Resource methods
- Entity provider for method arguments or return type

```
Representations - example
@ProduceMime({"application/xml", "application/json"})
public interface ObjectResource {
            @GET
            public Artist getObjects();
            @GET
            @ProduceMime("text/html")
            public String getObjectAsHtml();
            ...
}
```

JAX-RS load Entity Manager

- At the top of your JAX-RS Resource class reference the Entity Manager that us used to access the JPA Enitity

com.ibm.prism.rest.ObjectResource

```
public class ObjectResource {

    @Context
    private UriInfo uriInfo;

    private ObjectsManager dbManager;

    public ObjectResource() {
        dbManager = new ObjectsManager();
    }

  ....
```

JAX-RS GET with JPA

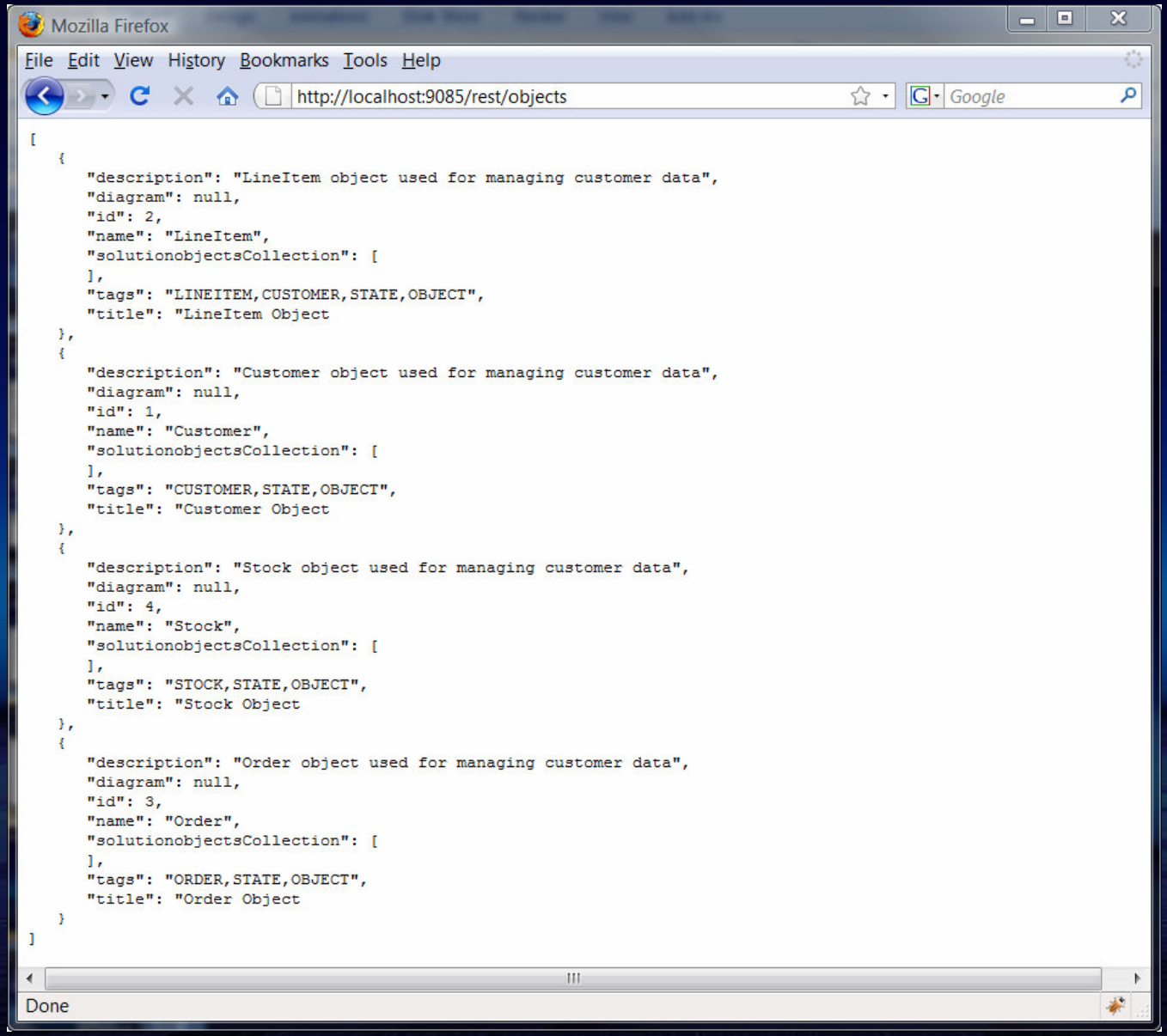
- Rest GET with JPA Query

com.ibm.prism.rest.ObjectResource

```
@Produces(value={"application/json"})
@GET
public Response get() {

    List<Objects> stores = dbManager.getObjects();
    Object json = null;
    Response resp =  Response.status(500).build();
    try
    {
    json = ObjectToJson.toJson(stores,null,true);
    String retjson = "";
    if (json instanceof JSONArtifact)
    {
       retjson = ((JSONArtifact) json).serialize(true);
       resp = Response.ok(retjson).build();
    }
    }
    catch(Exception e)
    {
       System.out.println(e.getStackTrace());
    }

    return resp;
}
```

Use Web 2.0 FEP **JSON4J** API to Transform Collection into JSON

Result



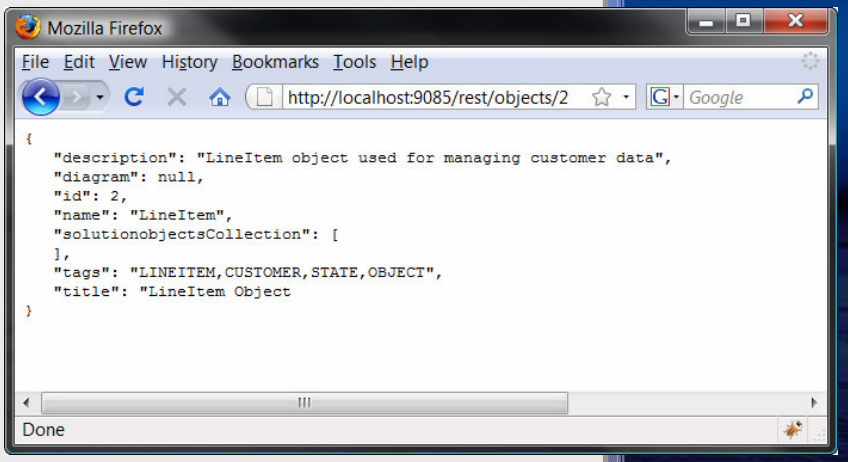

```
[
   {
      "description": "LineItem object used for managing customer data",
      "diagram": null,
      "id": 2,
      "name": "LineItem",
      "solutionobjectsCollection": [
      ],
      "tags": "LINEITEM,CUSTOMER,STATE,OBJECT",
      "title": "LineItem Object
   },
   {
      "description": "Customer object used for managing customer data",
      "diagram": null,
      "id": 1,
      "name": "Customer",
      "solutionobjectsCollection": [
      ],
      "tags": "CUSTOMER,STATE,OBJECT",
      "title": "Customer Object
   },
   {
      "description": "Stock object used for managing customer data",
      "diagram": null,
      "id": 4,
      "name": "Stock",
      "solutionobjectsCollection": [
      ],
      "tags": "STOCK,STATE,OBJECT",
      "title": "Stock Object
   },
   {
      "description": "Order object used for managing customer data",
      "diagram": null,
      "id": 3,
      "name": "Order",
      "solutionobjectsCollection": [
      ],
      "tags": "ORDER,STATE,OBJECT",
      "title": "Order Object
   }
]
```

JAX-RS GET a single item with JPA

com.ibm.prism.rest.ObjectResource

```
@Consumes("text/xml")
@Produces(value={"application/json"})
@GET
@Path(value="/{objectid}")
public Response get(@PathParam(value="objectid") String objectid)
{
    Response resp =  Response.status(500).build();

 try
     {
        int id = Integer.parseInt(objectid);
        Objects object = dbManager.findObjectsById(id);


     Object json = ObjectToJson.toJson(object,null,true);
     String retjson = "";
     if (json instanceof JSONArtifact)
     {
            retjson = ((JSONArtifact) json).serialize(true);
            resp = Response.ok(retjson).build();

     }

     }
     catch(Exception e)
     {
        System.out.println(e.getStackTrace());
     }

     return resp;
}
```

Taking objectid from entity request we find the object and return it serialized in JSON



JAX-RS DELETE with JPA

com.ibm.prism.rest.ObjectResource

```
@Consumes("text/xml")
@DELETE
@Path(value="/{objectid}")
public Response delete(@PathParam(value="objectid") String objectid) {

int id = -1;
try
{
    // Parse id into Int and then try and remove the Object
    id = Integer.parseInt(objectid);
    Objects objdel = new Objects();
    objdel.setId(id);
        dbManager.deleteObjects(objdel);

    return Response.status(204).build();

}
catch ( Exception exc)
{
return Response.status(500).build();
}

}
```

Using the Object ID we call the Entity Manager API to delete the record

JAX-RS CREATE with JPA

com.ibm.prism.rest.ObjectResource

```
@Consumes("application/json")
@POST
public Response post(Objects object) {


        Response resp = Response.ok().build();
        try
        {
            resp = Response.ok().build();
            dbManager.createObjects(object);
            resp.getMetadata().putSingle("Name", "/" + object.getName());
        }
        catch (Exception e)
        {
        resp = Response.status(500).build();
        }

        return resp;

}
```

Return the unique ID created for the record

JAX-RS UPDATE with JPA

com.ibm.prism.rest.ObjectResource

```
@Consumes("application/json")
@PUT
@Path(value="/{objectid}")
public Response put(Objects object, @PathParam(value="objectid") String objid) {
    Response resp = null;
    try
    {
           int id = Integer.parseInt(objid);
           Objects obj = new Objects();
           obj.setId(id);
           String updated = dbManager.updateObjects(obj);
           resp = Response.status(204).build();

           if(updated.equals("")) {
               resp.getMetadata().putSingle("Name", "/" + obj.getName());
           }

    }
    catch(Exception e)
    {
            resp = Response.status(500).build();
    }

    return resp;
}
```

What is not covered in JAX-RS

- REST is an architecture, JAX-RS a possible implementation in Java
  - JAX-RS is bound to HTTP
- No description, registration and discovery (You can use WADL or WSDL)
- Very little media types (representations)
- Connectedness, caching, partial-GET must be implemented by developer
- No Client APIs that is left to XHR

PROS

- Simplicity
- Uniformity
- Different representations
- Cacheable
- Scalable
- Connectedness

CONS

- Not transport protocol independent JMS, HTTP, Email,…
- No standards for 2-phase commit transactions
- WS-AtomicTransaction
- No standards for security except for https
- Signatures, encryption

Conclusion

- SOAP is excellent Computer to Computer Protocol
- With JAX-RS, REST will even be a more serious alternative for SOAP.
- Only a few situations thinkable where SOAP is the only alternative.
- It is possible to use SOAP and REST together, but when using one, stick to it.

- REST and JAX-RS offers a very easy and consumer able interface to enterprise information

- Moving JEE server into a Data Application Server to support Mashups and the next generation of Web 2.0 solutions.

- JAX-RS and JPA working together makes data integration with Web 2.0 very easy and flexible.

- IBM is releasing with HP a JAX-RS implementation to Apache in March 2009
- Tools will be coming !

GET CODING !!!

Questions

- Please remind attendees to complete their session evaluation form
- Thank you!!