



ORACLE

**A Persistence Journey:
From XML to the Database and Back Again**

Mike Keith
michael.keith@oracle.com

About Me

- Almost 20 years experience in server-side and persistence implementations
- Java and persistence architect at Oracle
- Active member of JCP expert groups, including JPA 2.0 and Java EE 6
- Co-wrote Pro EJB 3: Java Persistence API
- Contributor to various specification and expert groups within JCP, OASIS and OSGi Alliance
- Presenter at numerous conferences and events

ORACLE

About You

- How many people have used or know enough to use JPA?
- How many people have used or know enough to use JAXB?
- How many people know what SDO is?
- How many people have ever used SDO?
- How many people are tired of me asking questions?

ORACLE

What will you learn?

- **What** the Eclipse Persistence Services Project is
- **Why** this project exists, and why you should care
- **Where** the project is situated in the context of the entire Eclipse ecosystem
- **When** the project can be downloaded and used
- **How** this project can be used for persistence of every kind, and how you can benefit from it
- **Who** is involved (and how you can be too!)

ORACLE

Stuff We Might Talk About

- Overview of the EclipseLink Project
- XML Mapping Problem
- Additional XML Support
- Object-Relational Mapping Problem
- Java Persistence API
- Putting it together
- Service Data Objects
- The Weather

ORACLE

What is Eclipse?

- Eclipse is an open source community
- Eclipse is more than just an IDE
 - Equinox (OSGi)
 - Rich Client Platform (RCP)
 - Higgins (Trust Framework),
 - Eclipse Communication Framework (ECF)
 - Swordfish (SOA Framework)
- Now...
 - Server-side Eclipse = Eclipse Runtime (RT)

ORACLE

Eclipse RT

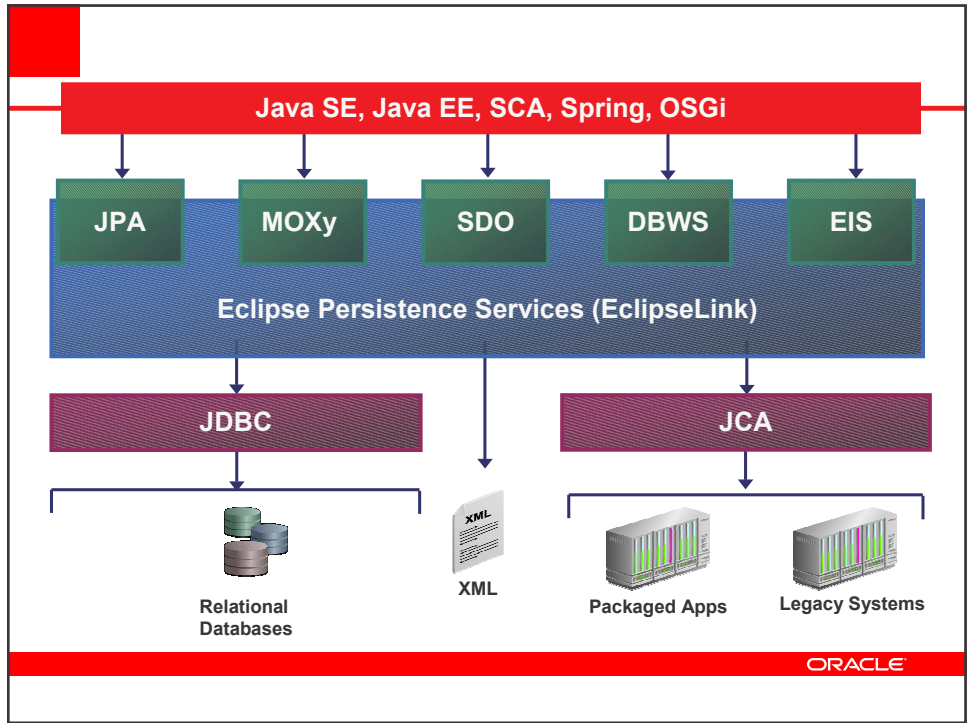
- Eclipse Communication Framework
- Equinox (OSGi)
- embedded Rich Client Platform (RCP)
- **Eclipse Persistence Services Project**
- Rich Ajax Platform (RAP)
- Riena Platform Project
- SMILA (SeMantic Info Logistics Architecture)
- Swordfish (SOA Framework)

ORACLE

Eclipse Persistence Services

- Nicknamed “EclipseLink”
- Comprehensive persistence
 - Eclipse JPA: Object-Relational
 - Eclipse MOXy: Object-XML
 - Eclipse SDO: Service Data Objects
 - Eclipse DBWS: Database Web Services
 - Eclipse EIS: Non-Relational using JCA
- Currently active in defining blueprints for OSGi persistence services
- Can be used standalone, or integrated with any runtime container or application server

ORACLE



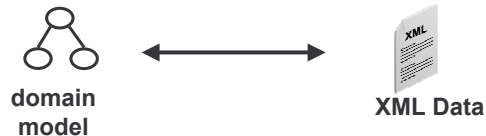
Eclipse JPA

- JPA 1.0 compliant implementation
- JPA 2.0 features under development
- Java EE, Java SE, Web, Spring, and OSGi
- Any JDBC/SQL compliant database
- Extensible and pluggable
- Huge number of feature extensions

The diagram shows a domain model (represented by a tree structure) on the left and a relational database (represented by a stack of cylinders) on the right. A double-headed arrow connects them, indicating bidirectional mapping. The Oracle logo is at the bottom right.

Eclipse MOXy

- Provides complete Object-XML mapping
 - Allows developers to work with XML as objects
 - Efficiently produce and consume XML
 - Document Preservation
- Supports Object-XML standard - JAXB
 - Provides additional flexibility to allow complete control on how objects are mapped



ORACLE

Eclipse MOXy Benefits

- Rich set of mappings providing complete control and flexibility to map objects to any XSD
 - Direct, composite object, composite collection, inheritance, positional, path, transformation
- Development Approaches
 - Model + Annotations → XSD
 - XSD → Model + Annotations
 - Model + Mappings(Annotations or XML)
- Supports any JAXP compliant parser
 - SAX, DOM, StAX
- Visual Mapping support using Workbench

ORACLE

Eclipse SDO

- What can you do?
 - Marshall/Unmarshall objects to/from XML
 - Define Types/Properties programmatically or derive from XSD
 - Generate JavaBean classes from XSD
 - Advanced mapping support for greater flexibility
- Why would you use it?
 - Schema/Structure unknown at compile time
 - Declarative metadata based tools/frameworks
 - XML-centric applications, need open content support
 - Dynamic content user interfaces

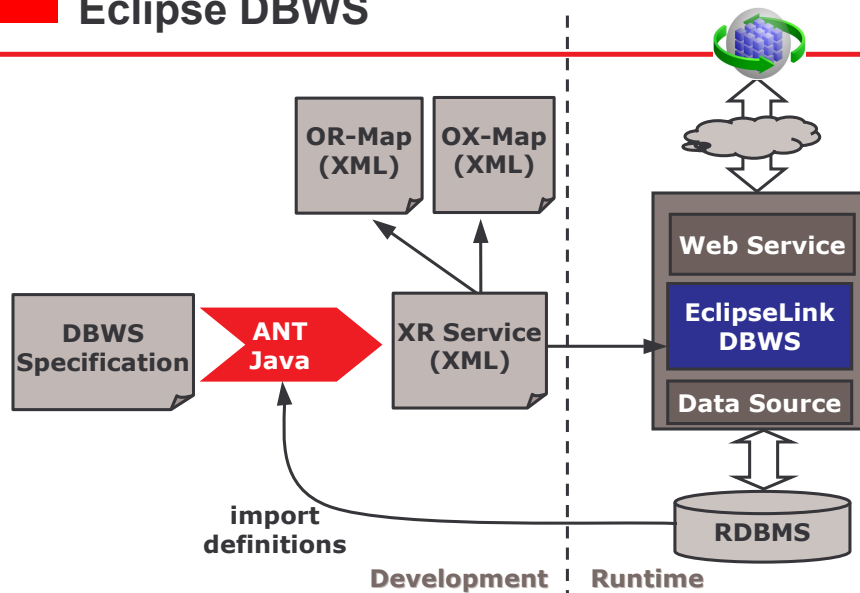
ORACLE

Eclipse DBWS

- Simplified and efficient access to relational data through Web Services
- Relational-to-XML data translation
- Minimal configuration with dev utilities to retrieve metadata and generate/package Web Service
- Developers can fully customize the database access and XML mapping of the data
- Ideal for usage within SOA/SCA

ORACLE

Eclipse DBWS



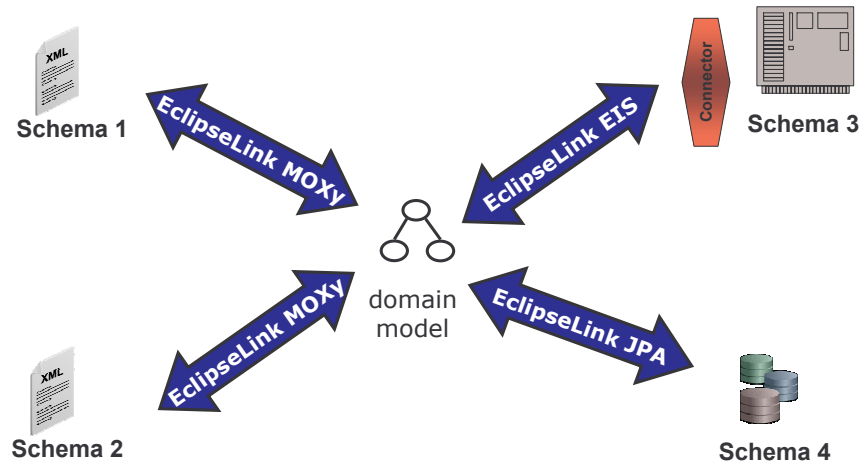
ORACLE

Eclipse EIS

- Provide persistence support for non-relational data stores using Java EE Connector Architecture (JCA)
- Mapping interaction inputs and outputs to persistent domain model
 - XML mapping leveraging Eclipse MOXy
 - Common Client Interface (CCI) mapping
- Visual mapping Workbench support
- Out of the box support for:
 - MQSeries, OracleAQ, Sun JCA, XML Files

ORACLE

Common Domain Model



ORACLE

Whirlwind XML Review

- XML schema document (XSD) defines rules for a specific kind of XML document instance
- XML instances are composed of **elements**, **attributes**, **text**
- Structure can be accessed using XPath
 - Nested element paths using "/"
 - Predicates can go in "[]"
 - Index of element using ".<indexNum>" notation
 - Optional @ for property traversal

ORACLE

Service Data Objects (SDO)

- Service Component Architecture
 - Created in OSOA consortium, now in OASIS
 - Architecture for SOA
 - Pluggable, configurable components
- SDO designed to be the currency of SCA
 - Not coupled to it, though
- Converted to XML for transfer
- Static and dynamic mode APIs
- Designed for disconnected access/manipulation

ORACLE

Data Access Service (DAS)

- Service responsible for retrieving/storing DO's
- Not supposed to be coupled to storage type
- Not currently specified or standardized
- Every current DAS implementation is proprietary
- No agreement yet on how implemented
 - single generic API
 - API for each storage type
- Standard attempted in OSOA
- Opinions?

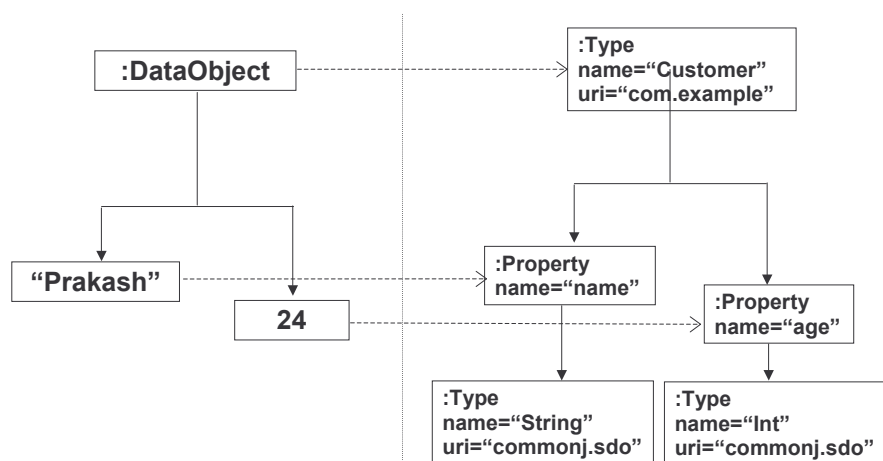
ORACLE

Data Objects

- Interface exposed to client is **DataObject**
- DataObject (DO) is of a given (obtainable) **Type**
- DO's have a series of typed **Property** objects
- Each Type has attributes, e.g.
 - **open** – can add properties dynamically
 - **sequenced** – properties are ordered
- Properties may be accessed using:
 - Reflection-based API
 - Property index
 - Path-based access using Xpath

ORACLE

Data Object Model



ORACLE

Data Graph

- A DO may be related to other DO's
- Tree of DOs form a **data graph**
- Containment – every DO except the root must be contained by another DO in the graph
- After obtaining data graph it may be manipulated
- Each data graph stores a **change summary**
 - Internally managed, publicly accessible
 - Keeps track of disconnected modifications
 - May be used by DAS at graph storage-time

ORACLE

Helper classes

- Loads of “Helper” classes to help do stuff
- Main one is the **HelperContext**
 - Provides access to all of the others
 - Defines a metadata scope

```
public interface HelperContext {
    CopyHelper getCopyHelper();
    DataFactory getDataFactory();
    DataHelper getDataHelper();
    EqualityHelper getEqualityHelper();
    TypeHelper getTypeHelper();
    XMLHelper getXMLHelper();
    XSDHelper getXSDHelper();
}
```

ORACLE

Using Helper classes

```
// Create helpers
HelperContext ctx = HelperProvider.getDefaultContext();
XSDHelper xsdHelper = ctx.getXSDHelper();
XMLHelper xmlHelper = ctx.getXMLHelper();

// Load in the types from the schema
schema = new FileInputStream("Customer.xsd");
List types= xsdHelper.define(schema, null);
schema.close();

// Load a sample XML file into DataObjects
input = new FileInputStream("input.xml");
XMLDocument xmlDoc = xmlHelper.load(input);
input.close();
DataObject obj = xmlDoc.getRootObject();
...
```

ORACLE

Static Data Objects

- Types/Properties can be generated from schema
- Client-facing set of interfaces and impl classes
- Interfaces are not coupled to SDO
- Classes are known beforehand
 - Applications can compile against them
 - Code completion, etc.
 - Stronger typing

ORACLE

Demo

Generating Static SDO's

ORACLE

Using Static Data Objects

```
...  
  
// Read in the data objects  
DataObject root = xmlDoc.getRootObject();  
  
// Assume the root element is of our root type  
CustomerType cust = (CustomerType) root;  
  
// Do a bunch of stuff using regular Java typed API's  
cust.getPersonalInfo().getLastName();  
ContactInfo cInfo = cust.getContactInfo();  
List phones = cInfo.getPhoneNumber(); // no generics!  
  
// Remove the first phone number  
phones.remove(0);  
...
```

ORACLE

Creating Dynamic Data Objects

```
// Access the type dynamically
Type customerType = TypeHelper.INSTANCE.getType(
    "http://www.example.com/customer",
    "customer-type");
// Create a customer DO based on the type
DataObject custDO =
    DataFactory.INSTANCE.create(customerType);
// Create related DO's
DataObject cInfoDO =
    custDO.createDataObject("contact-info");
DataObject bAddrDO =
    cInfoDO.createDataObject("billing-address");
```

ORACLE

Using Dynamic Data Objects

```
DataObject cust = doc.getRootObject();

// Get the billing address
DataObject billingAddrDO =
    cust.getDataObject("contactInfo/billingAddress");

// Get the personal info
DataObject pInfoDO =
    cust.getDataObject("personalInfo");

// Get the first phone number and last name
DataObject phone =
    pInfoDO.getDataObject("phoneNumber.0");
String lastName = pInfoDO.getString("lastName");
```

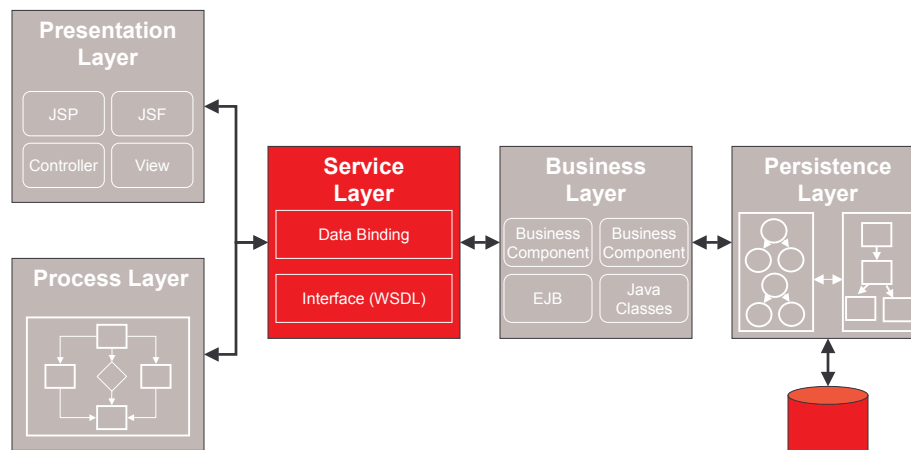
ORACLE

XML Mapping – What is it Good For?

- Configuration files
 - Read and store XML files, but manipulate in Java
- Interoperable data storage
 - Simple requirements, non-RDBMS
- Interoperable data exchange
 - B2B - Pre-arranged schema, common infrastructure
 - Web Services – Standard WSDL, different infrastructure
- Systems Integration
 - Internal or external

ORACLE

The Wonderful World of SOA



ORACLE

JAX-WS 2.1

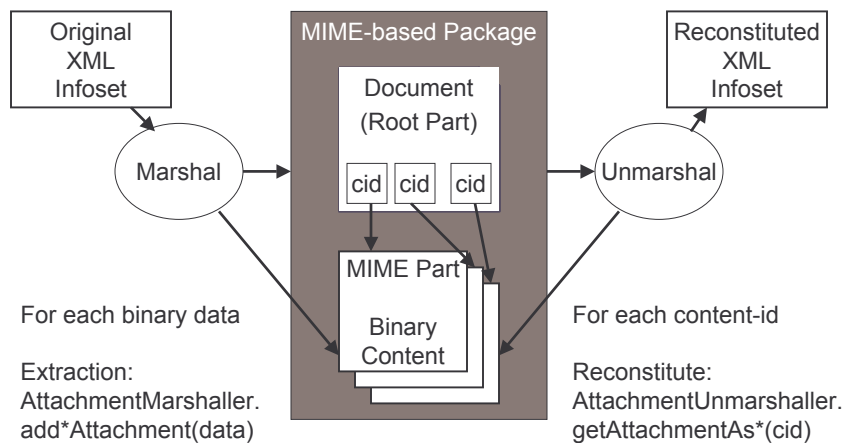
Java API for XML-Based Web Services

- Successor to JAX-RPC 1.1
- Uses JAXB 2.1 as the XML Binding Layer
- JAXB 2.1 compiler included as of Java SE 6 (04)
- Binding Support for WS Attachments
 - SOAP (MTOM)
 - XML-binary Optimized Packaging (XOP)
 - WS-I Attachment Profile 1.0 (WSIAP)

ORACLE

JAX-WS 2.1 – Binary Attachments

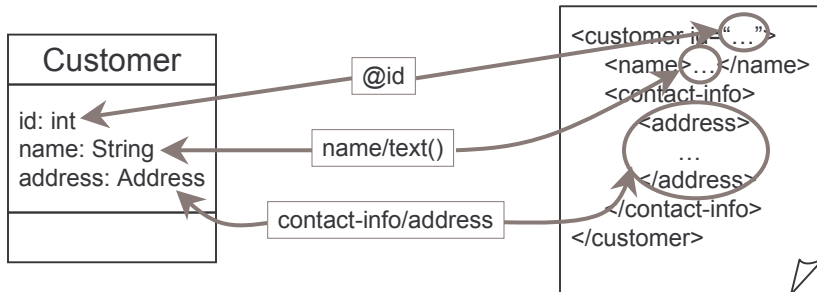
JAXB Marshal/Unmarshalling of Binary Attachments



ORACLE

The XML Mapping Problem

The activity of **mapping** is the process of connecting objects/attributes to XML types/nodes by XPath



ORACLE

Transparent XML Access

- We really want the XML to be transparent from the business logic perspective
 - Manual Coding of an XML conversion layer
 - Need to manually fill in objects using DOM model
 - Web Services
 - Accessed in whatever form the web service is written
 - JAX-WS uses XML binding to generate WSDL
 - Business Objects
 - Accessed as objects or components
 - Need binding layer in middle to handle the object-XML mapping and conversion

ORACLE

Manual Layer (“Old School”)

- Use XML parser directly
- Introduction of JAXP at least made the parser standard!
- Manage, invoke and manipulate DOM nodes and/or SAX/StAX events
- End up embedding element names in processing and conversion code
- “Grungy”, monotonous and error-prone
- Really have 2 Java models for the same data

ORACLE

Old School Example

Ex. Read in customer and get the first name

```
try {
    factory = DocumentBuilderFactory.newInstance();
    builder = factory.newDocumentBuilder();
    document = builder.parse(new File("custRecord.xml"));
} catch (Exception ex) { ... }

Element custElement = document.getDocumentElement();
Node child = custElement.getFirstChild();
while(child != null) {
    if(child.getNodeName().equals("first-name")) {
        Node textNode = child.getFirstChild();
        firstName = textNode.getNodeValue();
        return firstName;
    }
    child = child.getNextSibling();
}
```

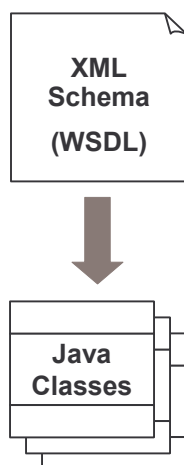
ORACLE

Much Better to Have a Standard

- Why keep re-inventing the DOM wheel and writing boilerplate code?
- The problem is exactly the same, it's just the concrete schemas that change
- Shared knowledge and experience
- Multiple implementations with additional features provided by vendors
- As XML became ubiquitous, the need became more apparent
- Finally introduced by JSR 31 (2001/2002)
- Now at version 2.1, and JAXB RI is even shipped as part of JDK 6

ORACLE

Use Case #1 – Top Down



- Start with an XML Schema (or WSDL)
- Generate an object model according to a set of rules
- Need support for all of the weird and wonderful XML Schema constructs
- Typically a static generation stage so that the domain classes can be used by programs
- Each change to the schema requires regeneration of the object model
- Set of rules may need to be large to meet the reqts of Java developers

ORACLE

JAXB

- JAXB is the standard for binding XML schema to Java classes
- Originally intended to solve the top-down approach, but also supports bottom up (becoming more popular)
- Was slow to be adopted by users (mostly because it was not very convenient)
- Is now becoming a first class citizen
 - Part of Java SE
 - Used by JAX-WS for web services data binding
 - Marketed as part of the web services stack
- Is generally useful in many mapping scenarios

ORACLE

JAXB Bindings

- Bindings specified as schema elements
- Included in the actual schema document (uggh!) or in an external bindings file
- Inlined bindings wrapped in <annotation> elements:

```
<xsd:complexType name="CustomerType">
  <xsd:annotation>
    <xsd:appinfo>
      <jxb:class name="Customer">
        ...
      </jxb:class>
    </xsd:appinfo>
  </xsd:annotation>
  ...
</xsd:complexType>
```

ORACLE

Storing Bindings Externally

- An external binding file allows binding information to be independent from schema file

```
<jxb:bindings schemaLocation = "customer.xsd"
  node = "xs:schema">
  <jxb:bindings
    node="//xs:complexType[@name='addressType']">
    <jxb:class name="Address"/>
    <jxb:bindings node="//xs:element[@name='zip']">
      <jxb:property name="province"/>
    </jxb:bindings>
  </jxb:bindings>
  ...
</jxb:bindings>
```

ORACLE

Domain Model Generation

- **xjc** compiler used to compile a schema with its bindings
- XML Schema instance is passed in and Java classes get spit out the other end
xjc [- options] <schemaFile>
- Resulting Java classes should be instantiated by invoking generated ObjectFactory class
- Lots of additional flags and options to control package name, and other preferences

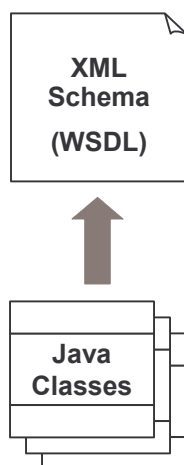
ORACLE

Demo

Using xjc

ORACLE

Use Case #2 – Bottom Up



- Start with an object model
- Generate an XML Schema (or WSDL) according to a set of rules
- Need support for all of the weird and wonderful XML Schema constructs
- May be invoked statically or dynamically
- Each change to the object model requires regeneration of the schema
- Set of rules may need to be extensive to generate specific required schema

ORACLE

Binding/Mapping Annotations

- JAXB 2.0 introduced annotations to the mix
- Provide a better basis for bottom-up schema generation, as well as binding info for runtime API
- **Package** maps to an XML target namespace
- **Type** maps to an XML schema type
 - *Class* maps to a complex type
 - Primitives and wrappers map to simple types
- **Field** or **property** maps to an attribute or an embedded element
- **Enum type** maps to simple schema type with restrictions

ORACLE

Binding/Mapping Annotations

- Default rules make the mapping experience more palatable
- Annotations are all prefixed with **Xml**, and [mostly] defined in **javax.xml.bind.annotation** package
- Can be specified at package (package-info.java), class, enum type, field/property
- XML is hierarchical – there exists the notion of a root element (**XmlRootElement**)
- Fields and properties are not ordered in Java, but order matters in XML

ORACLE

Binding/Mapping Annotations

```
@XmlElement
@XmlType(propOrder={"billingAddress","shippingAddress"})
public class Customer {
    @XmlAttribute(name="id")
    public int getId() {...}
    public void setId(int id) {...}

    @XmlElement(name="shipping-address")
    public Address getShippingAddress() {...}
    public void setShippingAddress(Address addr) {...}

    @XmlElement(name="billing-address")
    public Address getBillingAddress() {...}
    public void setBillingAddress(Address addr) {...}
}
```

ORACLE

Generating the XML Schema

- **Schemagen** tool used to generate a schema from a set of classes
- `schemagen [- options] <files>`
- Classes may be passed in source form or compiled class form
- Schema created will be a result of the mappings
- Currently no way to specify the name of the schema (there is an ant task, though)

ORACLE

Demo

Using schemagen

ORACLE

JAXB Implementation

- JAXB implementation is determined by the factory that is used to create the **JAXBContext**
- Factory class is determined by the contents of the **jaxb.properties** file
- So, to use the EclipseLink context factory, the jaxb.properties file needs to look like this:

```
javax.xml.bind.context.factory= \  
    org.eclipse.persistence.jaxb.JAXBContextFactory
```

ORACLE

JAXB Runtime - JAXBContext

- **JAXBContext** is the client API entry point
- Acts as a factory for:
 - Marshallers and Unmarshallers
 - JAXBIntrospectors
 - Binders
- Create a new instance, initializing with context information that is either:
 - Context path(s)
 - Classes
- May also pass in a context path classloader

ORACLE

Initializing the Context

Load metadata from annotations:

```
Class[] classes = new Class[3];
classes[0] = org.example.Customer.class;
classes[1] = org.example.Address.class;
classes[2] = org.example.PhoneNumber.class;
JAXBContext jaxbContext =
    JAXBContext.newInstance(classes);
```

Load externalized metadata:

```
JAXBContext jaxbContext =
    JAXBContext.newInstance("com.customer");
```

ORACLE

Unmarshalling the Data

- Use an Unmarshaller instance obtained from JAXBContext
- Can unmarshal from a number of different sources, including streams, a file, a URL, DOM, SAX, etc.
- May enable or disable default validation
- May define listeners and event handlers that get triggered during the unmarshalling process

```
File file = new File("input.xml");
Unmarshaller unmarshaller = ctx.createUnmarshaller();
Customer customer =
    (Customer) unmarshaller.unmarshal(file);
```

ORACLE

Marshalling the Data

- Use a Marshaller instance obtained from JAXBContext
- Can marshal to a number of different targets, including streams, a SAX content handler, a DOM
- May enable or disable default validation
- May define listeners and event handlers that get triggered during the unmarshalling process

```
Marshaller marshaller = ctx.createMarshaller();
marshaller.marshal(customer, System.out);
```

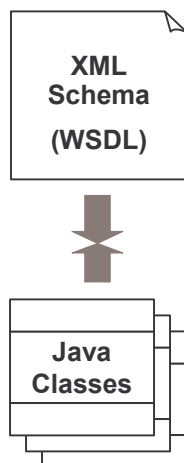
ORACLE

Demo

JAXB Demo

ORACLE

Use Case #3 – Meet in the Middle



- Start with an object model and an existing XML Schema (or WSDL)
- Map the constructs to each other
- Currently there is no suitable standard for doing this
- Mapping information is independent of both sides
- Changes to one will not require regeneration or changes to the other

ORACLE

Parser Independence

Write Once, Run Anywhere

- Most XML binding layers are bound to a specific version of an XML parser.
- Most enterprise applications are run on application servers.
- Each application includes and depends on a specific version of an XML parser.
- If the binding layer and application server are dependent on different XML parsers, then it may be impossible to use them together.

ORACLE

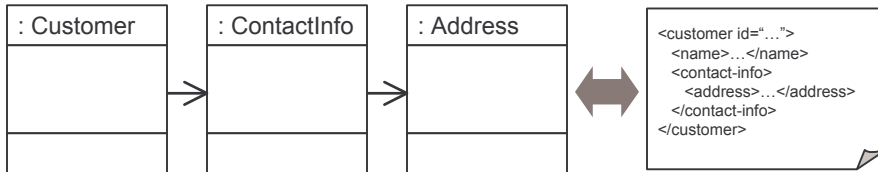
What If my Model is too Hard to Map?

- **Converters**
Convert datatypes between your object model and XML document.
- **Adapters**
Unable to map an object to XML. An adapter allows you to convert your object to an object that can be mapped.
- **Events**
Have access to your object at various stages before and after it is marshalled/unmarshalled. Also access to the unmapped content of your XML document.

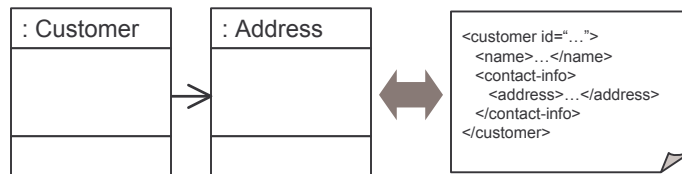
ORACLE

The Object Model

Generated Object Model



Your Domain Objects



ORACLE

External Mapping Metadata

Map using an XML file

- Mapping information is kept in XML descriptors and not in the objects.
- Metadata can be version controlled independent of what it is mapping
- External metadata means this approach is NOT intrusive on either the object model or the XML schema.
- The object model can be mapped to multiple XML representations.

ORACLE

EclipseLink Workbench

- Shipped with EclipseLink as a utility
- Provides convenient graphical environment to map objects to XML
- Standalone and written in Java
 - runs in any environment
- Stores mapping/runtime information in portable, deployable XML file
- Also supports ORM and EIS projects

ORACLE

Demo

Mapping with MOXy

ORACLE

Java Persistence API (JPA)

- Persistence API for operating on POJO **entities**
- Merger of expertise from TopLink, Hibernate, JDO, EJB vendors and individuals
- Created as part of EJB 3.0 within JSR 220
- Released May 2006 as part of Java EE 5
- Integration with Java EE web and EJB containers provides enterprise “ease of use” features
- “Bootstrap API” can also be used in Java SE
- Pluggable Container-Provider SPI

ORACLE

Anatomy of an Entity

- Abstract or concrete top level Java class
 - Non-`final` fields/properties, no-arg constructor
- No required interfaces
 - No required business or callback interfaces (but you may use them if you want to)
- Direct field or property-based access
 - Getter/setter can contain logic (e.g. for validation)
- May be `Serializable`, but not required
 - Only needed if passed by value (in a remote call)

ORACLE

A Simple Entity

- Must be indicated as an Entity (`@Entity`)
- Must have a persistent identifier (primary key)

```
@Entity
public class Employee {
    @Id int id;

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
}
```

ORACLE

Persistent Identity

- Identifier (id) in entity, primary key in database
- Uniquely identifies entity in memory and in db

1. Simple id – single field/property
`@Id int id;`
 2. Compound id – multiple fields/properties
`@Id int id;`
`@Id String name;`
 3. Embedded id – single field of PK class type
`@EmbeddedId EmployeePK id;`
- Uses
PK
class

ORACLE

Persistent Context

- Abstraction representing a set of **“managed”** entity instances
 - Entities keyed by their persistent identity
 - Only one entity with a given persistent identity may exist in the PC
 - Persistent entities not referenced by PC are **“detached”**
- Controlled and managed by EntityManager
 - Contents of PC change as a result of operations on EntityManager API

ORACLE

Entity Manager

- Client-visible artifact for operating on entities
 - API for all the basic persistence operations
- Can think of it as a proxy to a persistence context
 - May access multiple different persistence contexts throughout its lifetime
- Multi-dimensionality leads to different aspects of EntityManager (and persistence context) naming
 - Transaction type, life cycle

ORACLE

Operations on Entities

- EntityManager API:
 - `persist()` - Insert the state of an entity into the db
 - `remove()` - Delete the entity state from the db
 - `refresh()` - Reload the entity state from the db
 - `merge()` - Synchronize state of detached entity with the pc
 - `find()` - Execute a simple PK query
 - `createQuery()` - Create query instance dynamically
 - `createNamedQuery()` - Create instance for predefined query
 - `createNativeQuery()` - Create instance for SQL query
 - `contains()` - Determine if entity is managed by pc
 - `flush()` - Force synchronization of pc to database

ORACLE

`persist()`

- Insert a new entity instance into the database
- Save the persistent state of the entity and any owned relationship references
- Entity instance becomes managed

```
public Customer createCustomer(int id, String name) {
    Customer cust = new Customer(id, name);
    entityManager.persist(cust);
    return cust;
}
```

ORACLE

find() and remove()

- find()
 - Obtain a managed entity instance with a given persistent identity – return null if not found
- remove()
 - Delete a managed entity with the given persistent identity from the database

```
public void removeCustomer(Long custId) {  
    Customer cust =  
        entityManager.find(Customer.class, custId);  
    entityManager.remove(cust);  
}
```

ORACLE

merge()

- State of detached entity gets merged into a managed copy of the detached entity
- Managed entity that is returned has a different Java identity than the detached entity

```
public Customer storeUpdatedCustomer(Customer cust) {  
    return entityManager.merge(cust);  
}
```

ORACLE

Queries

- Dynamic or statically defined (**named queries**)
- Criteria using **JP QL** (extension of EJB QL)
- Native SQL support (when required)
- Named parameters bound at execution time
- Pagination and ability to restrict size of result
- Single/multiple-entity results, data projections
- Bulk update and delete operation on an entity
- Standard hooks for vendor-specific hints

ORACLE

Operations on Queries

- Query instances are obtained from factory methods on EntityManager
- Query API:
 - `getResultList()` – execute query returning multiple results
 - `getSingleResult()` – execute query returning single result
 - `executeUpdate()` – execute bulk update or delete
 - `setFirstResult()` – set the first result to retrieve
 - `setMaxResults()` – set the maximum number of results to retrieve
 - `setParameter()` – bind a value to a named or positional parameter
 - `setHint()` – apply a vendor-specific hint to the query
 - `setFlushMode()` – apply a flush mode to the query when it gets run

ORACLE

Dynamic Queries

- Use `createQuery()` factory method at runtime and pass in the JP QL query string
- Use correct execution method
 - `getResultList()`, `getSingleResult()`, `executeUpdate()`
- Query may be compiled/checked at creation time or when executed
- Maximal flexibility for query definition and execution

ORACLE

Dynamic Queries

```
public List findAll(String entityName) {  
    return entityManager.createQuery(  
        "select e from " + entityName + " e")  
        .setMaxResults(100)  
        .getResultList();  
}
```

- Return all instances of the given entity type
- JP QL string composed from entity type. For example, if "Account" was passed in then JP QL string would be: **"select e from Account e"**

ORACLE

Named Queries

- Use `createNamedQuery()` factory method at runtime and pass in the query name
- Query must have already been statically defined either in an annotation or XML
- Query names are “globally” scoped
- Provider has opportunity to precompile the queries and return errors at deployment time
- Can include parameters and hints in static query definition

ORACLE

Named Queries

```
@NamedQuery (name="Sale.findByCustId",
             query="select s from Sale s
                   where s.customer.id = :custId
                   order by s.salesDate")
public List findSalesByCustomer(Customer cust) {
    return
        entityManager.createNamedQuery(
                        "Sale.findByCustId")
            .setParameter("custId", cust.getId())
            .getResultList();
}
```

- Return all sales for a given customer
- Use a named parameter to specify customer id

ORACLE

Object/Relational Mapping

- Map persistent object state to relational database
- Map relationships to other entities
- Metadata may be annotations or XML (or both)
- Annotations
 - Logical—object model (e.g. @OneToMany)
 - Physical—DB tables and columns (e.g. @Table)
- XML
 - Can additionally specify scoped settings or defaults
- Standard rules for default db table/column names

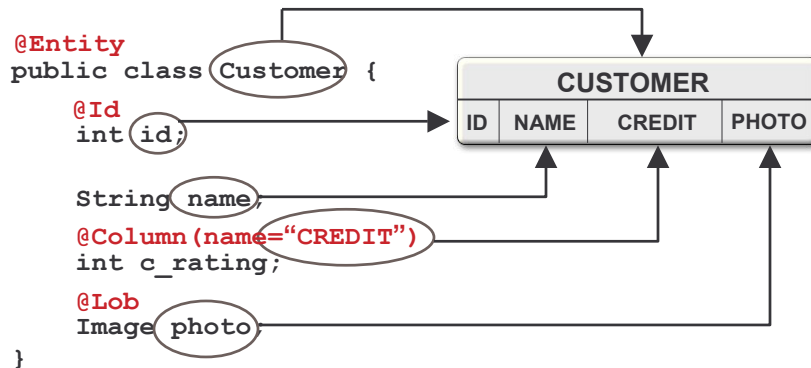
ORACLE

Simple Mappings

- Direct mappings of fields/properties to columns
 - @Basic - optional annotation to indicate simple mapped attribute
- Maps any of the common simple Java types
 - Primitives, wrappers, enumerated, serializable, etc.
- Used in conjunction with @Column
- Defaults to the type deemed most appropriate if no mapping annotation is present
- Can override any of the defaults

ORACLE

Simple Mappings



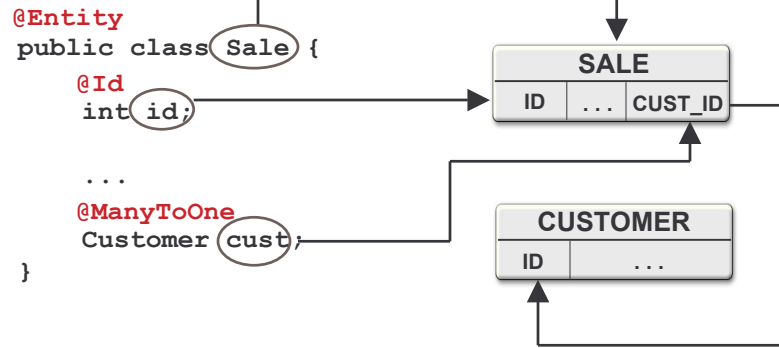
ORACLE

Relationship Mappings

- Common relationship mappings supported
 - `@ManyToOne`, `@OneToOne`—single entity
 - `@OneToMany`, `@ManyToMany`—collection of entities
- Unidirectional or bidirectional
- Owning and inverse sides of every bidirectional relationship
- Owning side specifies the physical mapping
 - `@JoinColumn` to specify foreign key column
 - `@JoinTable` decouples physical relationship mappings from entity tables

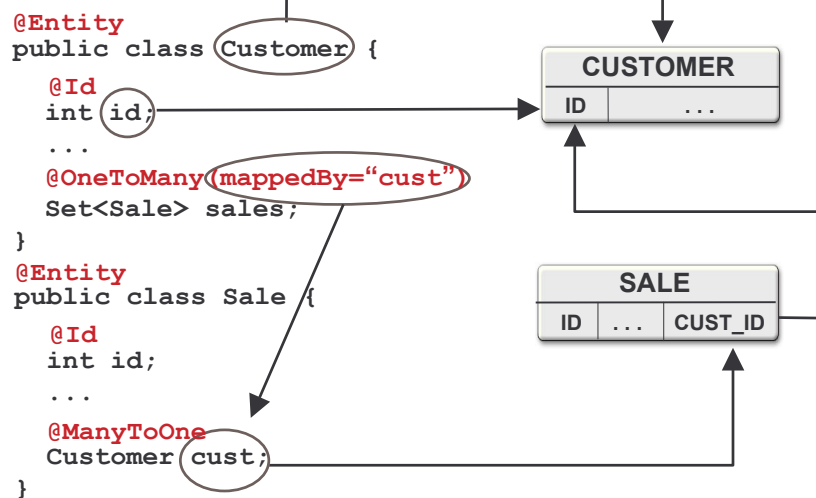
ORACLE

ManyToOne Mapping



ORACLE

OneToMany Mapping



ORACLE

Persistence in Java SE

- No deployment phase
 - Application must use a “**Bootstrap API**” to obtain an EntityManagerFactory
- Resource-local EntityManagers
 - Application uses a local **EntityTransaction** obtained from the EntityManager
- New application-managed persistence context for each and every EntityManager
 - No propagation of persistence contexts

ORACLE

Entity Transactions

- Only used by Resource-local EntityManagers
- Isolated from transactions in other EntityManagers
- Transaction demarcation under explicit application control using EntityTransaction API
 - **begin(), commit(), rollback(), isActive()**
- Underlying (JDBC) resources allocated by EntityManager as required

ORACLE

Bootstrap Classes

`javax.persistence.Persistence`

- Root class for bootstrapping an EntityManager
- Locates provider service for a named persistence unit
- Invokes on the provider to obtain an EntityManagerFactory

`javax.persistence.EntityManagerFactory`

- Creates EntityManagers for a named persistence unit or configuration

ORACLE

Example

```
public class PersistenceProgram {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence
            .createEntityManagerFactory("SomePUnit");
        EntityManager em = emf.createEntityManager();
        em.getTransaction().begin();
        // Perform finds, execute queries,
        // update entities, etc.
        em.getTransaction().commit();
        em.close();
        emf.close();
    }
}
```

ORACLE

EclipseLink JPA

- Acts as a JPA persistence provider
- Fully compliant support for JPA 1.0
- Will be Reference Implementation for JPA 2.0
- Oracle, Sun, plus other committers/contributors
- Not bound to any application server or environment
- Works on every compliant database with a supporting JDBC driver

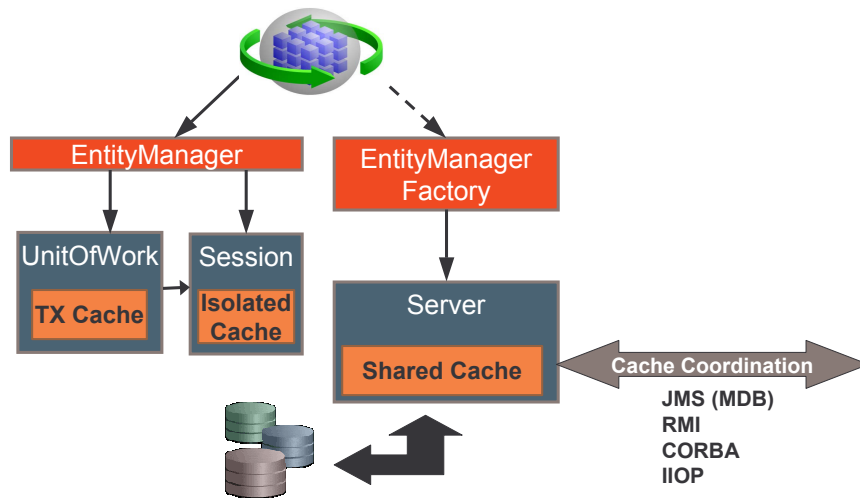
ORACLE

EclipseLink Caching

- Entity caching
 - L2 objects shared across transactions/users
 - Coordination in a clustered deployment
- Application specific configuration
 - Cache isolation: per client (EM) or shared
 - Cache Type and Size: Weak, Soft-Weak, Full, None
 - Expiration/Invalidation
 - Time to live, Time of day, API
 - Coordination (cluster-messaging)
 - Messaging: JMS, RMI, CORBA, RMI-IIOP, ...
 - Mode: SYNC, SYNC+NEW, INVALIDATE, NONE

ORACLE

Caching Architecture



Configuring the Cache

- Default: objects read are cached and trusted
- Configuration by entity type important
 - Volatility of data
 - Shared usage of data
- Configuration Parameters
 - Cache isolation, type, size, expiry, coordination
 - Refreshing
 - By query (use-case) or descriptor (always)
- Locking is the only way to avoid potential data inconsistency in concurrent write scenarios

ORACLE

Locking

- Prevent data corruption!
- Java Developers think of locking at the object level
- Databases may need to manage locking across many applications
- EclipseLink is able to respect and participate in locks at database level
 - Optimistic: Numeric, Timestamp, All fields, Selected fields, Changed field
 - Pessimistic

ORACLE

Query Framework

- Queries can be defined using
 - Entity Model: JPQL, Expressions, Query-by-example
 - Database: SQL, Stored Procedures
- Customizable
 - Locking, Cache Usage, Refreshing
 - Optimizations: Joining, Batching, parameter binding
 - Result shaping/conversions
- Static or Dynamic
 - Stored Procedure support

ORACLE

Eclipse JPA Extensions

- Extensions applied using annotations or XML
- Mappings
 - @BasicMap, @BasicCollection, @PrivateOwned, @JoinFetch
 - @Converter, @TypeConverter, @ObjectTypeConverter
- @Cache
 - type, size, isolated, expiry, refresh, cache usage, coordination
 - Cache usage and refresh query hints
- @NamedStoredProcedureQuery
 - IN / OUT / INOUT parameters, multiple cursor results

ORACLE

Eclipse JPA Extensions

- Locking
 - Non-intrusive policies @OptimisticLocking
 - Pessimistic query hints
- JDBC Connection Pooling
- Logging: Diagnostics, SQL, Debugging
- Weaving for lazy fetch and change tracking
 - Dynamic and Static
- Customization
 - Entity Descriptor: @Customizer, @ReadOnly
 - Session Customizer

ORACLE

Mapping Extensions

```
@Entity
@Cache(type=SOFT_WEAK, coordinationType=SEND_OBJECT_CHANGES)
@OptimisticLocking(type=CHANGED_COLUMNS)
@Converter(name="money", converterClass=MoneyConverter.class)
public class Employee {
    @Id
    private int id;

    private String name;

    @OneToMany(mappedBy="owner")
    @PrivateOwned
    private List<PhoneNumbers> phones;

    @Convert("money")
    private Money salary;
    ...
}
```

ORACLE

Database Platform

- Native SQL (dialect) support with custom operators
- Stored Procedures & Functions (output params)
- Extensible Advanced Data Types support
 - VPD/OLS and Proxy Authentication
- Configurable value return from write
- Default DB platform sensing
- Additional Oracle DB feature support
 - Virtual Private Database, Oracle Hints, Historical queries, Oracle Spatial, Object-relational structures, XMLDB, etc.

ORACLE

Server Platform

- Simplified configuration and mediator for host container environment
- Enables
 - Direct JTA integration
 - Data Source/JDBC connection un-wrapping
 - JMX MBean deployment
 - Logging integration
- Current Server Platforms
 - SunAS/GlassFish, OracleAS/OC4J, WLS, WAS, JBoss

ORACLE

Performance and Tuning

- Highly configurable and tunable
 - Principle: minimize and optimize database calls
 - Enable application specific tuning
- Flexibility allows efficient business models and relational schemas to be used
- Leverages underlying performance tuning features
 - Java, JDBC and the underlying database technology

ORACLE

Eclipse Dali

- IDE Tooling for JPA
- Sub-project of Web Tools Platform (WTP)
- Provides dev-time O-R mapping validation against relational database
- Annotation support and mapping assistance
- Persistence.xml descriptor configuration
- XML descriptor support

<http://www.eclipse.org/dali>

ORACLE

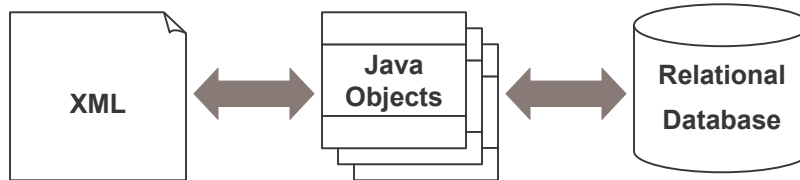
Demo

Mapping JPA with Dali

ORACLE

Combining JAXB and JPA

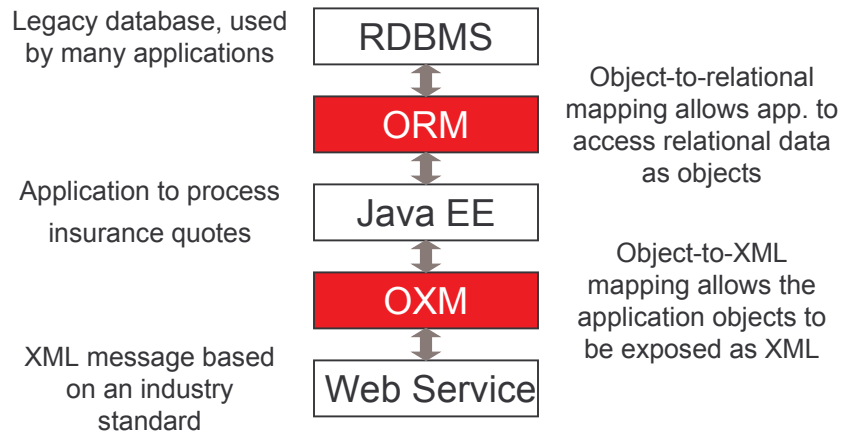
Java object may be a Java EE component, a Spring bean, or wherever your business logic is located



ORACLE

Scenario

One Object Model Mapped to RDBMS and XML



ORACLE

JAXB and JPA Mappings

```
@XmlRootElement
@Entity
public class Customer {

    @XmlAttribute(name="id")
    @Id
    public int getId() {...}
    public void setId(int id) {...}

    ...
    @XmlElement(name="billing-address")
    @OneToOne
    @JoinColumn(name="ADDR_ID")
    public Address getBillingAddress() {...}
    public void setBillingAddress(Address addr) {...}
}
```

ORACLE

Demo

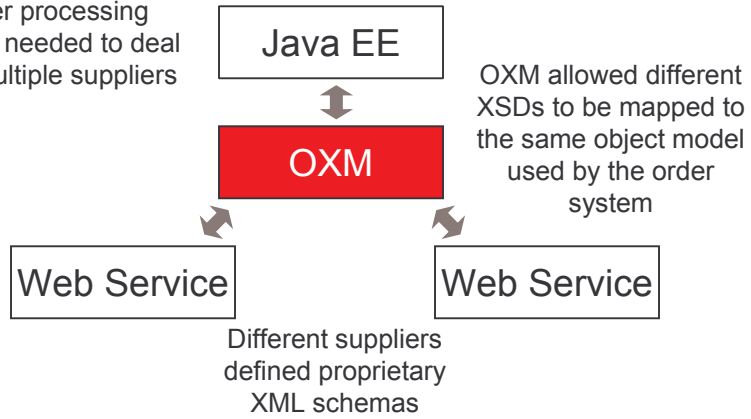
Combining JPA with JAXB

ORACLE

Scenario

One Object Model Mapped to Multiple XML Schemas

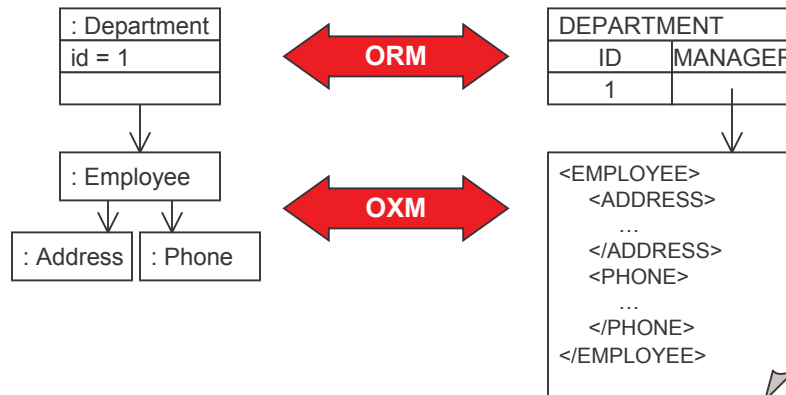
Order processing system needed to deal with multiple suppliers



ORACLE

Scenario

XML Data in your Database



ORACLE

EclipseLink Summary

- First comprehensive Open Source Persistence solution
 - Eclipse JPA: Object-Relational
 - Eclipse MOXy: Object-XML
 - Eclipse SDO: Service Data Objects
 - Eclipse DBWS: Database Web Services
 - Eclipse EIS: Non-Relational using JCA
- Mature and full featured
- Will be RI for JPA 2.0

ORACLE

Get Involved

- User
 - Try it out and provide feedback
 - File bug reports and feature requests
- Contributor
 - Contribute to roadmap discussions
 - Suggest bug fixes
- Committer
 - Committer base can still grow
 - Interested and capable individuals welcome!

ORACLE

More Information

Project:

www.eclipse.org/eclipselink

Newsgroup:

eclipse.technology.eclipselink

Wiki:

wiki.eclipse.org/index.php/EclipseLink

Committer Team Blog:

eclipselink.blogspot.com

ORACLE