Introducing EclipseLink

JA-SIG Conference, April 2008

Doug Clarke
douglas.clarke@oracle.com

ORACLE

ORACLE

A little about me

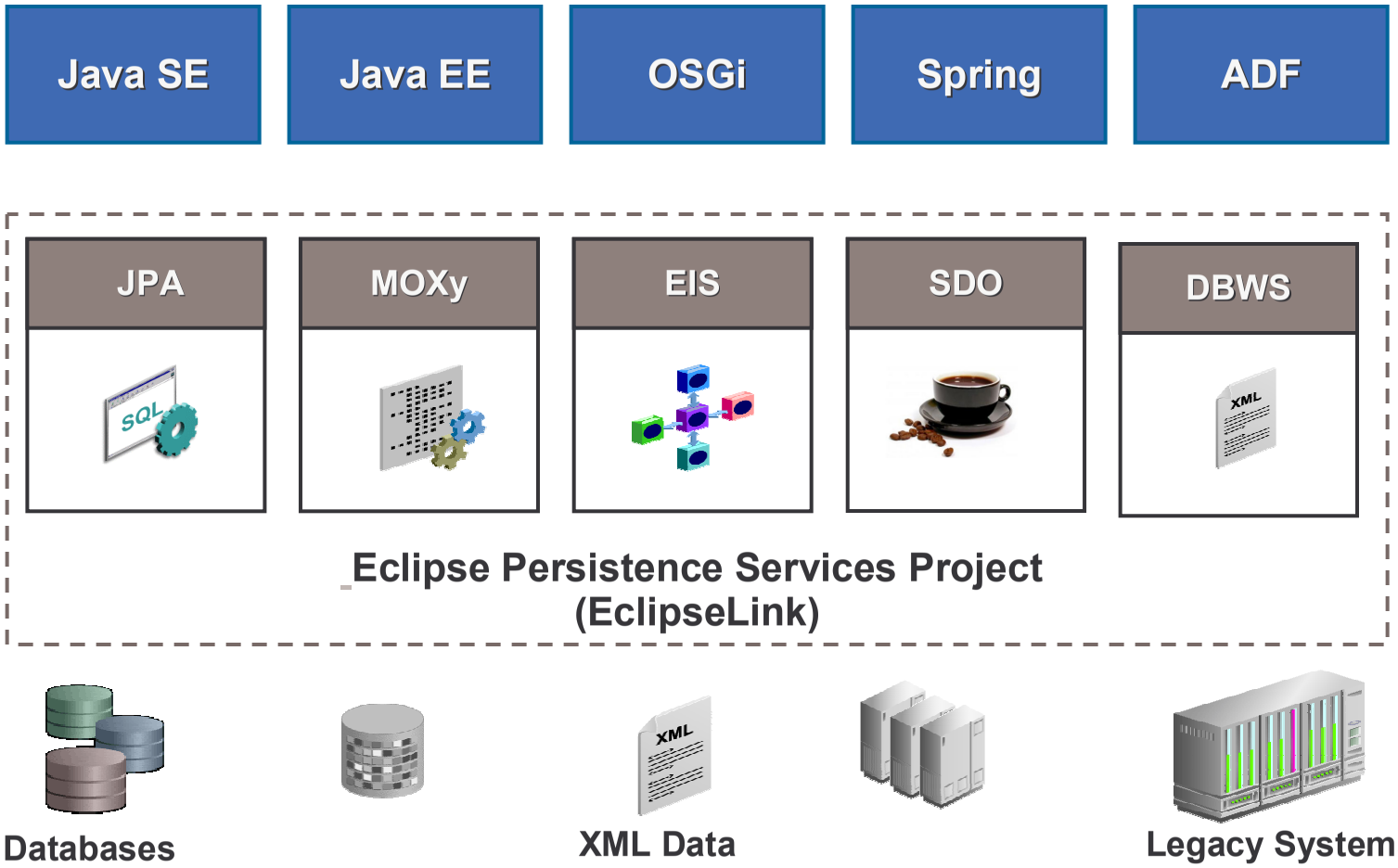
- Doug Clarke
  - Eclipse Persistence Services Project (EclipseLink)
    - Project co-Lead
  - Director of Product Management for Oracle TopLink
    - Involved with persistence technology for over 10 years

ORACLE

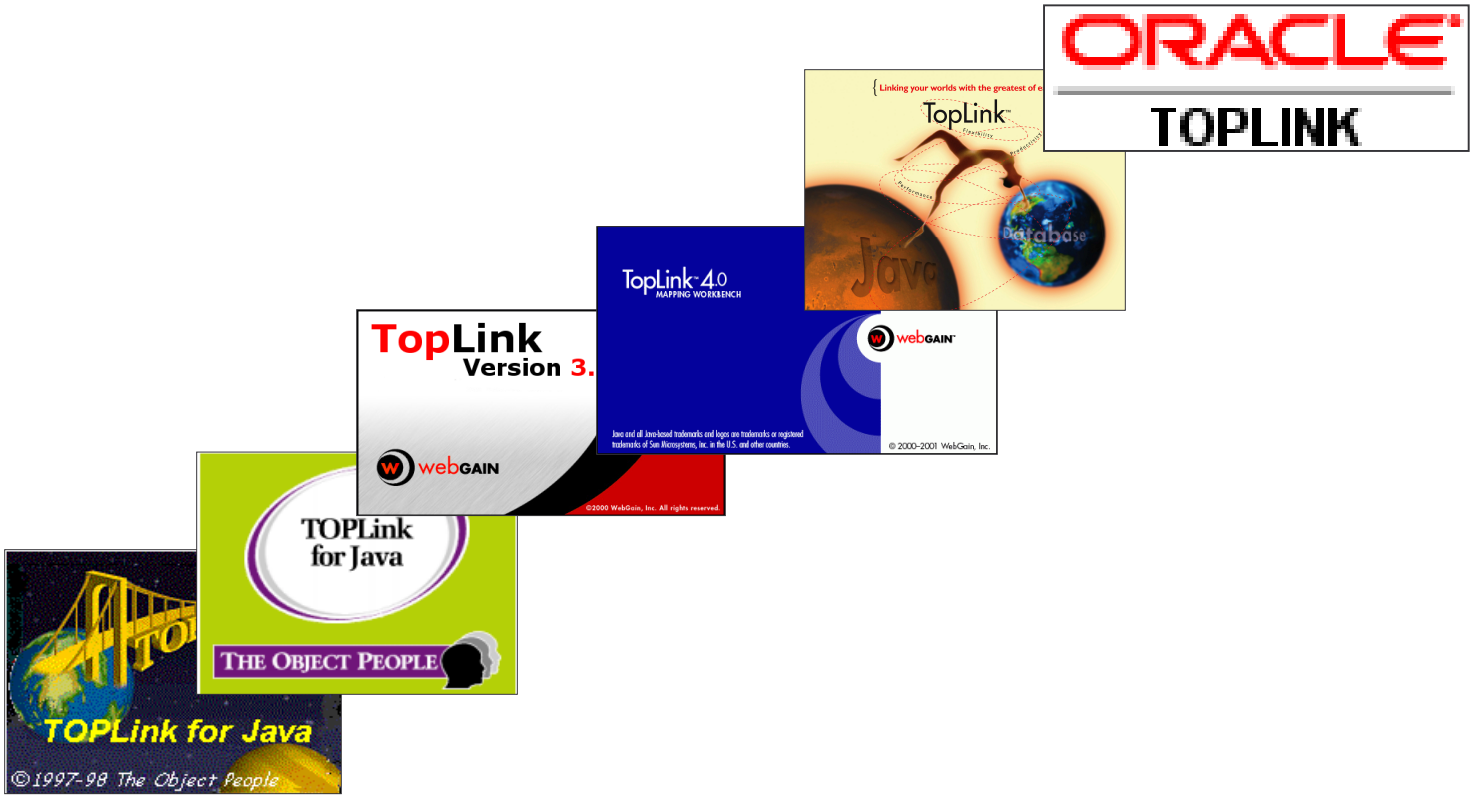
Eclipse Persistence Services

- Eclipse runtime project
  - Nicknamed “EclipseLink”
  - Currently Incubating in Technology Project
- Comprehensive
  - EclipseLink JPA: Object-Relational
  - EclipseLink MOXy: Object-XML
  - EclipseLink SDO: Service Data Objects
  - EclipseLink DBWS: Database Web Services
  - EclipseLink EIS: Non-Relational using JCA
  - Support for Java SE, Java EE, OSGi, and Spring
- Open Source, Open Standards, Advanced Features

Eclipse Persistence Services – *"EclipseLink"*



History of EclipseLink



1996 → 2007

ORACLE

Significance

- First comprehensive open source persistence solution
  - Object-Relational and much more
- Based upon product with 12 years of commercial usage
- Shared infrastructure
  - Easily share the same domain model with multiple persistence technologies
  - Leverage metadata for multiple services
- Important part of the Eclipse Ecosystem

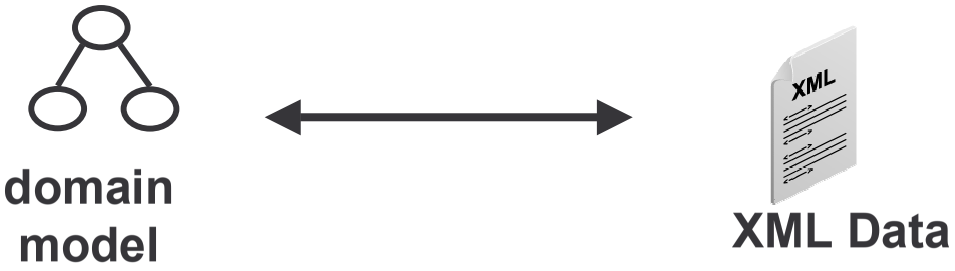
ORACLE

EclipseLink JPA

- JPA 1.0 compliant implementation
- Java EE, Java SE, Web, Spring, and OSGi
- Any JDBC/SQL compliant database
  - Advanced database extensions: Stored procedures, Native SQL
- Highly Extensible
- Schema generation
- Key infrastructure: Mapping, Querying, Transactions, Caching
- … plus many valuable advanced features

EclipseLink 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

EclipseLink 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

EclipseLink and OSGi

- Work with OSGi expert group to define OSGi persistence services blueprint
- Deliver EclipseLink as OSGi bundle(s)
- Show through examples how to leverage within an OSGi solution
- Address technical challenges as a community
- Current Status
  - OSGi Branch contains working examples: JPA, MOXy, and SDO
  - Porting changes into 1.0 (PDE projects)
  - Only requires Equinox specific functionality for weaving

ORACLE

Challenge: XML Development

- With rapid adoption of SOA and Web Services, XML has become pervasive
- XML is an ideal data exchange format, but is difficult to develop with directly
  - Requires complex, cumbersome code
  - Couples application logic to specific XML structure
  - Difficult to maintain

ORACLE

Java Access of XML Data

- Direct JAXP
  - Window on data
  - Direct use of an XML parser, uses DOM nodes and/or SAX/StAX events directly.
- Domain Objects/Entities
  - Accessed as objects or components (EJBs), transparent that the data is stored in XML
  - Need binding layer in middle tier to handle the object-XML mapping and conversion

Challenge: XML Development

Objective—obtain employee id

- JAXP

```
Node childNode = employeeElement.getFirstChild();
while(childNode != null) {
  if(childNode.getNodeName().equals("employee-id")) {
      Node employeeNumberTextNode = childNode().getFirstChild();
      employeeNumber = new
       Integer(employeeNumberTextNode.getNodeValue()).intValue();
  }
  childNode.getNextSibling();
}
```

- Using XML binding

```
employee.getId();
```

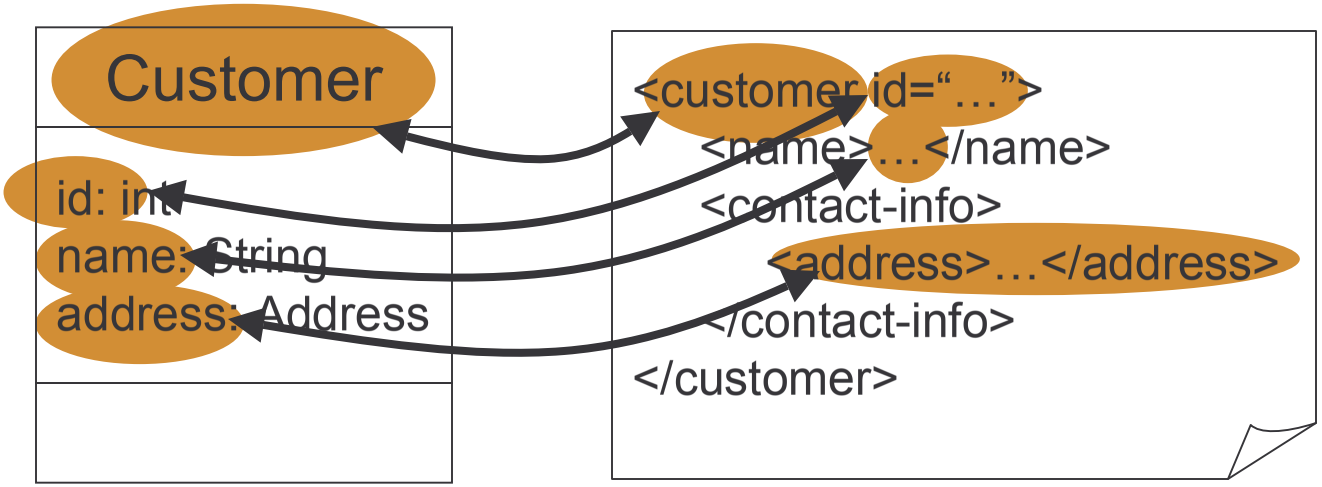

ORACLE

Data Binding Approaches

- Code Generation
- Declarative
  - Annotate Java Classes
  - Externalized Mapping Metadata

Data Binding/Mapping

- The activity of ‘Mapping’ is the process of connecting objects/attributes to XML types/nodes.



ORACLE

About Java Architecture for XML Binding (JAXB)

- JAXB 2 part of Java EE 5 specification
- Included in Java 6 SDK
- Suitable for use in different environments
  - Java SE, Java EE, OSGi, Spring
- A Java standard that defines:
  - how Java objects are converted to/from XML (specified using a standard set of mappings)
  - a programmer API for reading and writing Java objects to/from XML documents
  - a service provider interface (SPI) to allow for selection of JAXB implementation

ORACLE

JAXB 2 Goals (a subset)

1. Full W3C XML Schema support
2. Binding existing Java classes to generated XML schema
4. Ease of Development: Leverage J2SE 5.0 Language Extensions
8. Partial mapping of XML document relevant to application
11. Portability of JAXB mapped classes
15. Ease of Use - Manipulation of XML documents in Java

ORACLE

Features of JAXB 2

JAXB 2.0 Standardized on POJOs

- No binding logic in the generated classes.
- Metadata specified using Java annotations.
- The only compile time dependencies are standard JAXB classes and interfaces.
- Classes generated by one vendors compiler can be used in another vendors runtime.
- JAXB 2.0 compiler included in Java SE 6

JAXB Programmer API

```
// Instantiate the JAXB context.  The context path
// indicates which classes are involved in the XML binding
JAXBContext context =
  JAXBContext.newInstance("com.example.model");

// Unmarshal the objects from XML
File file = new File("input.xml");
Unmarshaller unmarshaller = context.createUnmarshaller();
Customer customer = (Customer)
  unmarshaller.unmarshal(file);

// Marshal the objects to XML
Marshaller marshaller = context.createMarshaller();
marshaller.marshal(customer, System.out);
```


ORACLE

JAXB 2—POJO Entities

- Concrete classes (POJOs)
- No required interfaces
- new() for instance creation
- Direct access or getter/setter methods
  - Can contain logic (e.g. for validation, etc.)

Mapping with Annotations on Fields

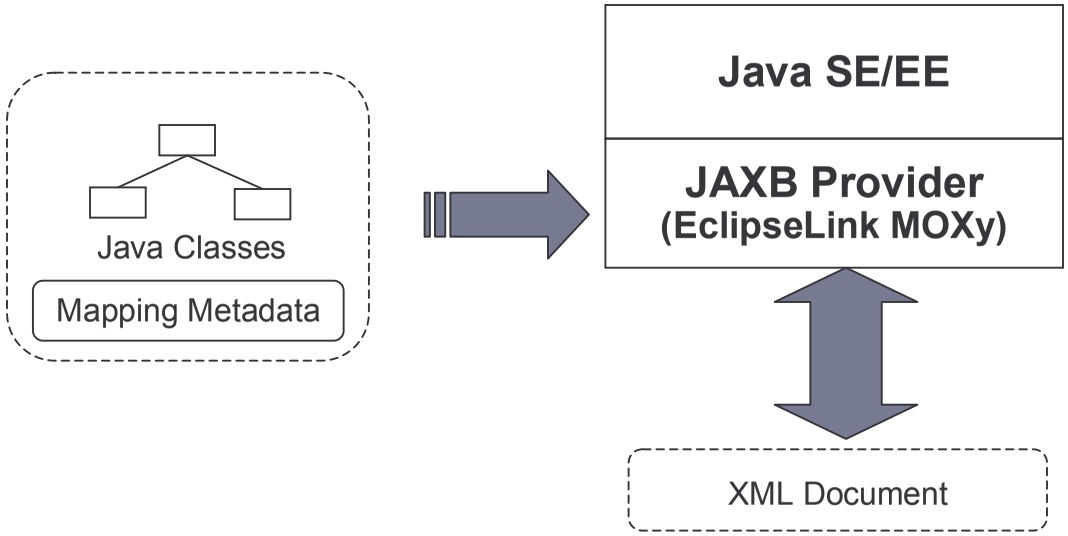
```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "customer-type", propOrder = {
    "firstName",
    "lastName",
    "billingAddress",
    "shippingAddress",
    "phoneNumber"
})
public class Customer {

    @XmlElement(name = "first-name", required = true)
    protected String firstName;
    @XmlElement(name = "last-name", required = true)
    protected String lastName;
    @XmlElement(name = "billing-address", required = true)
    protected Address billingAddress;
    @XmlElement(name = "shipping-address", required = true)
    protected Address shippingAddress;
    @XmlElement(name = "phone-number",
          namespace = "urn:customer-example", required = true)
    protected List<PhoneNumber> phoneNumbers;
```


Mapping with Annotations on Properties

```
@XmlAccessorType(XmlAccessType.PROPERTY)
@XmlType(name = "customer-type", propOrder = {
    "firstName",
    "lastName",
    "billingAddress",
    "shippingAddress",
    "phoneNumber"
})
public class Customer {

    protected String firstName;
    protected String lastName;
    protected Address billingAddress;
    protected Address shippingAddress;
    protected List<PhoneNumber> phoneNumbers;

    @XmlElement(name = "first-name", required = true)
    public String getFirstName() {
        return firstName;
    }
  ...
```


JAXB 2 Runtime

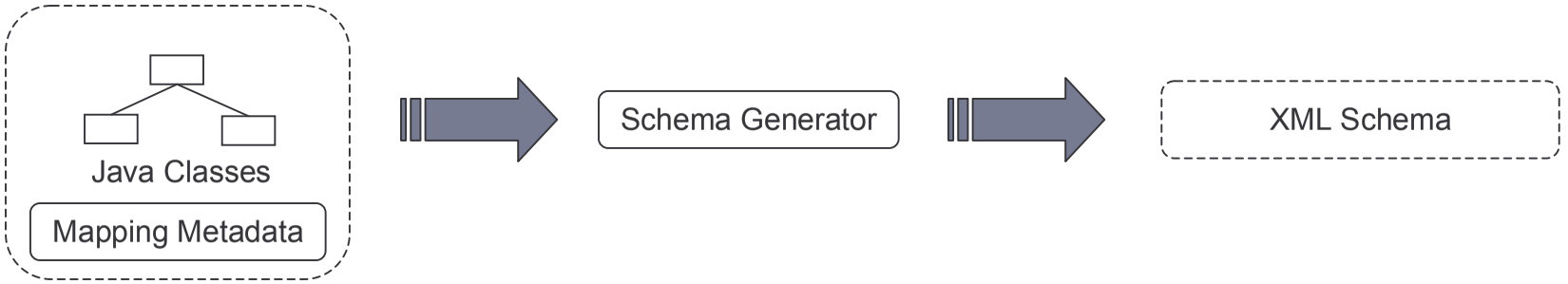
- JAXB runtime combines:
  - Java Classes
  - Mapping Metadata



JAXB 2 Design Time

- JAXB Schema Compiler:

- JAXB Schema Generator:



ORACLE

EclipseLink MOXy

"**M**apping **O**bjects to **X**ML"

- High performance JAXB 2.1 Implementation
- Mapping engine shared with EclipseLink JPA
- Provides support for standard Object/XML mapping technologies:
  - JAXB 2.1
    - Full compliance scheduled for 1.1 release
  - Supports XML mapping metadata
  - Used in EclipseLink SDO and DBWS

Java Persistence API (JPA)—in a Nutshell

- A Java standard that defines:
  - how Java objects are stored in relational databases (specified using a standard set of mappings)
  - a programmer API for reading, writing, and querying persistent Java objects (“Entities”)
  - a full featured query language
  - a container contract that supports plugging any JPA runtime in to any compliant container.

ORACLE

JPA—Background

- Separate document bundled as part of EJB 3.0 specification
- Suitable for use in different modes
  - Standalone in Java SE environment
  - Hosted within a Java EE Container
- Standardization of current persistence practices
- Merging of expertise from persistence vendors and communities including: TopLink, Hibernate, JDO, EJB vendors and individuals

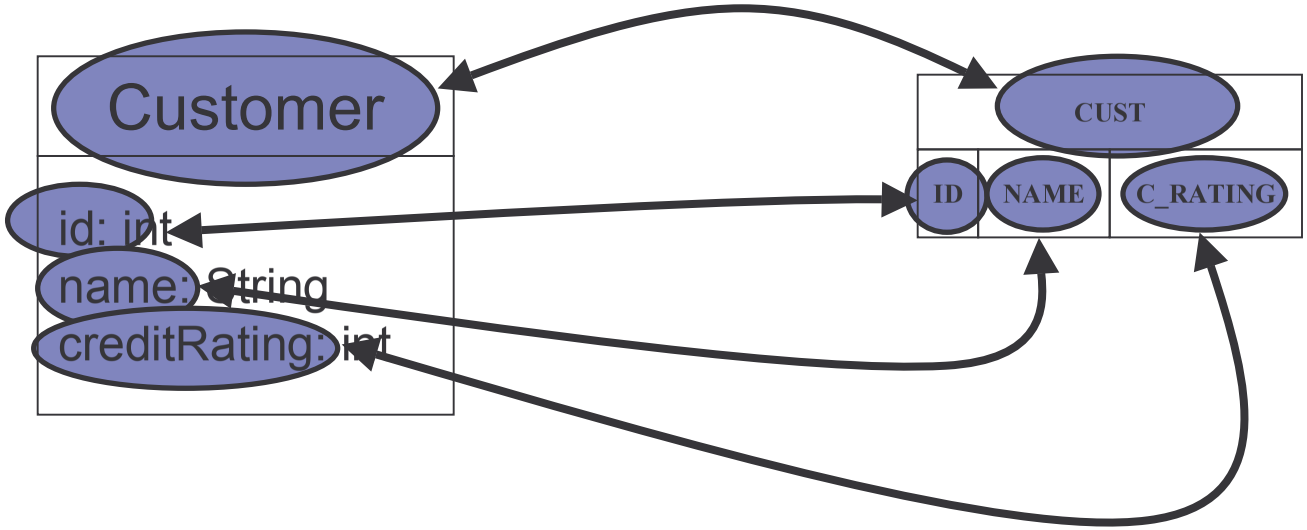
ORACLE

JPA—POJO Entities

- Concrete classes (POJOs)
- No required interfaces
  - No required business interfaces
  - No required callback interfaces
- new() for instance creation
- Direct access or getter/setter methods
  - Can contain logic (e.g. for validation, etc.)

Mapping

- The activity of ‘Mapping’ is the process of connecting objects/attributes to tables/columns



ORACLE

Object-Relational Mappings

- Core JPA Mappings
  - Id
  - Basic
  - Relationships
    - OneToOne
    - ManyToOne
    - OneToMany
    - ManyToMany
  - And more…
- Annotations and/or XML

Annotations on Fields

```
@Entity public class Customer {

    @Id
    private String name;
    @OneToOne
    private Account account;

    public String getName() { return name; }
    public void setName(String name) {
        this.name = name;
    }
    public Account getAccount() { return account; }
    public void setAccount(Account account) {
        this.account = account;
    }
}`
```

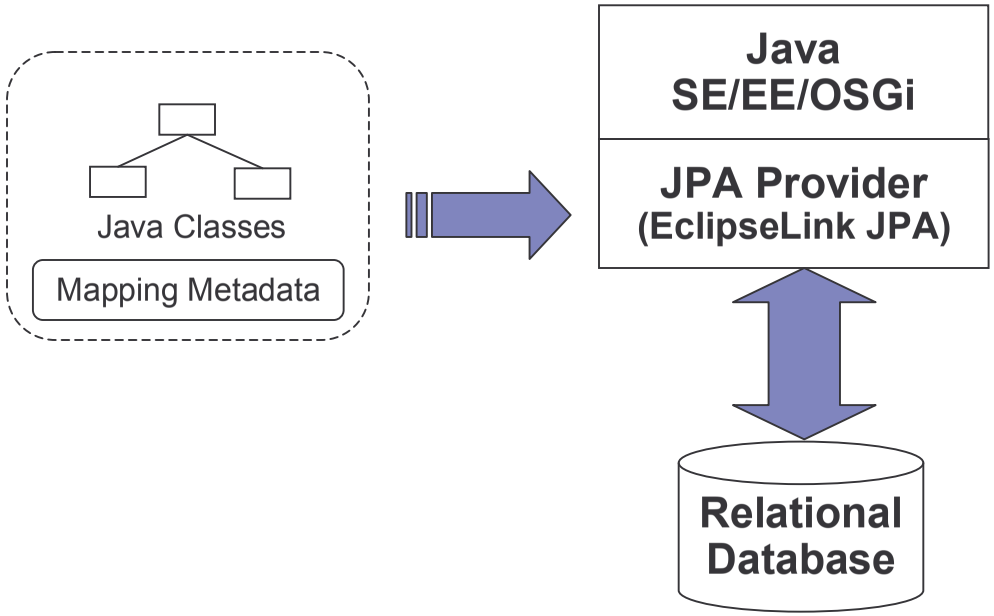

Annotations on Properties

```
@Entity public class Customer {

    private String name;
    private Account account;

    @Id
    public String getName() { return name; }
    public void setName(String name) {
        this.name = name;
    }
    @OneToOne
    public Account getAccount() { return account; }
    public void setAccount(Account account) {
        this.account = account;
    }
}
```


Mappings in XML

```
<entity-mappings
  xmlns="http://java.sun.com/xml/ns/persistence/orm"
…
   <entity class="Customer">
     <attributes>
       <id name="name"/>
       <one-to-one name="account"/>
     </attributes>
   </entity>
…
</entity-mappings>
```


JPA Runtime



ORACLE

EclipseLink JPA … Advanced Features

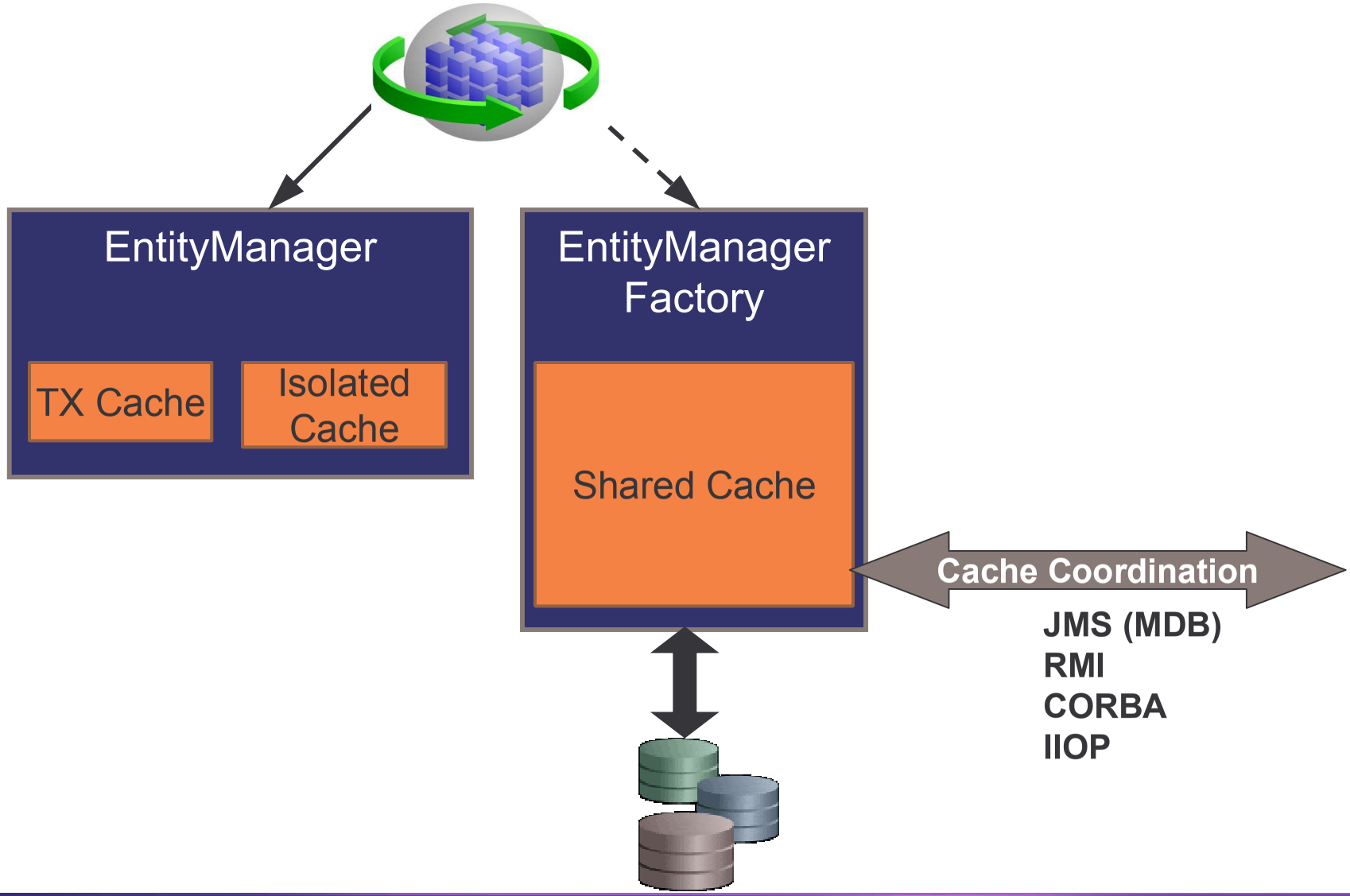
- Maintain the JPA configuration and programming model
- Expose extended functionality
  - Persistence Unit properties
  - Query hints
  - Custom annotations
  - Native API (minimize required usage)

ORACLE

Advanced EclipseLink JPA

- Advanced features supported through annotations and XML
- Mapping
  - @BasicMap, @BasicCollection
  - @PrivateOwned, @JoinFetch
  - @Converter, @TypeConverter, @ObjectTypeConverter
- @Cache
  - type, size, isolated, expiry, refresh, cache usage, coordination
  - Cache usage and refresh query hints

Caching Architecture



Cache Configuration Options

- Cache Invalidation/Expiration
  - Time to live
  - Fixed Times
  - Programmable (external notification)
- Shared and Isolated caching
- Cache Coordination
  - Messaging
    - JMS, RMI, IIOP, CORBA, OC4J-JGroups
  - Type specific configuration
    - Modes: Sync, Sync+New, Invalidate, None
- All configurable on a per type basis

ORACLE

More Advanced EclipseLink JPA

- @NamedStoredProcedureQuery
  - IN/OUT/INOUT parameters, multiple cursor results
- Locking
  - Non-intrusive policies @OptimisticLocking
    - ALL_COLUMNS, CHANGED_COLUMNS, SELECTED_COLUMNS, VERSION_COLUMN (@Version)
  - Pessimistic query hints
- JDBC Connection Pooling
- Logging: Diagnostics, SQL, Debugging
- Customization
  - Entity Descriptor: @Customizer, @ReadOnly
  - Session Customizer

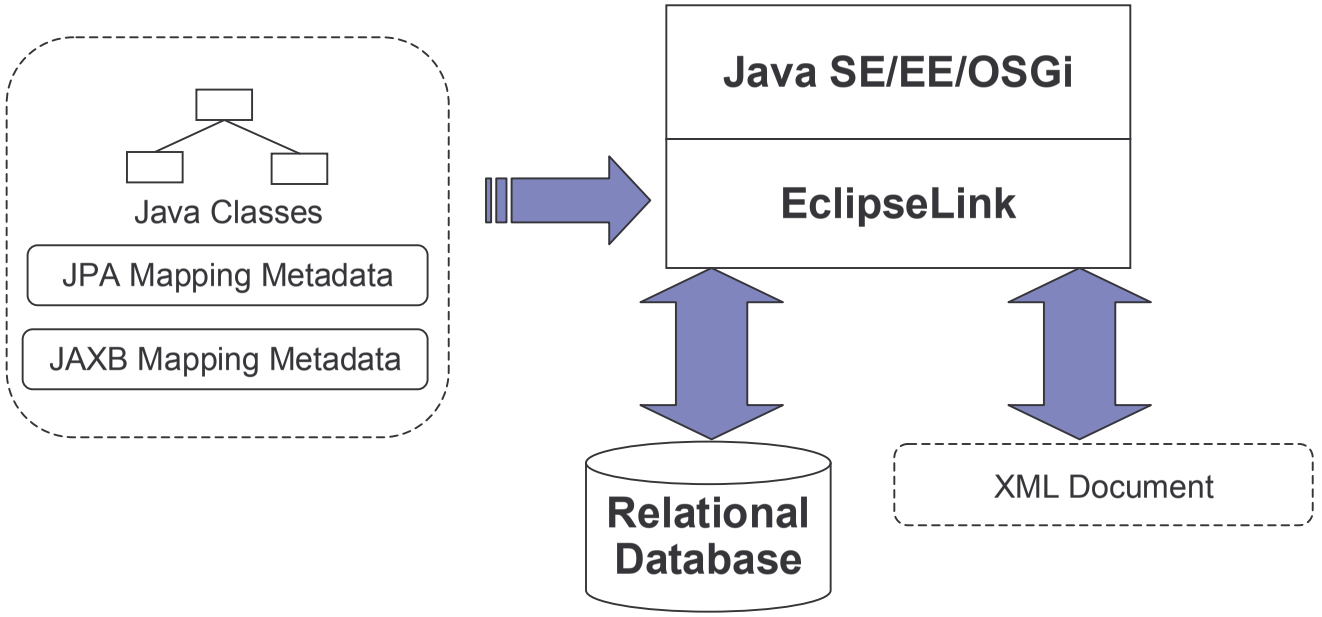
Example

```
@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
  …
}
```


Weaving Support

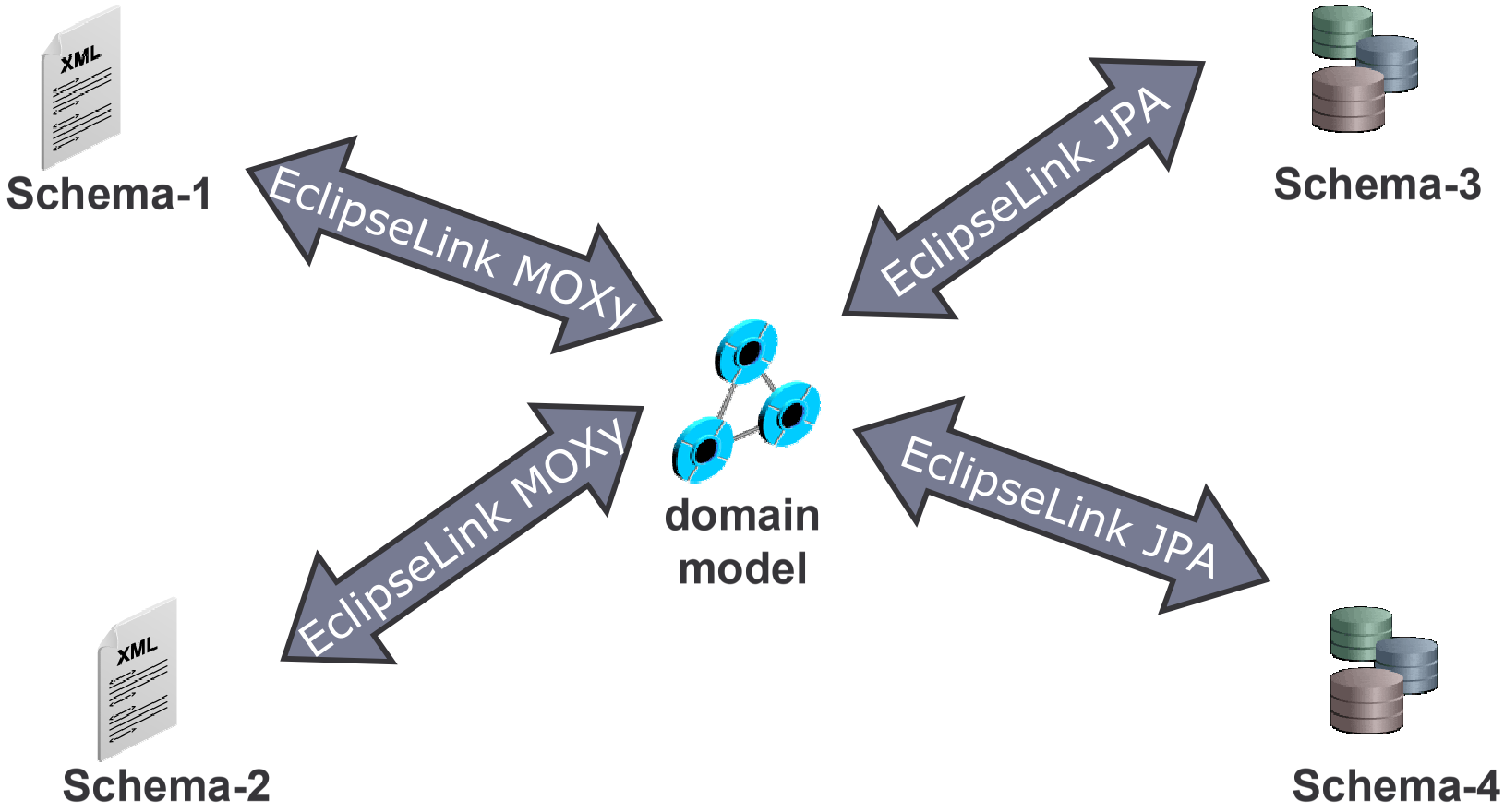
- EclipseLink makes use of Weaving (ASM) to introduce additional functionality into the JPA entity classes
  - Lazy Loading
  - Optimized Change Tracking
- Available for Java SE using JDK/JRE's –javaagent:
  - Optional
- Static weaving also supported
  - Weaving of .class files before deployment

Combining MOXy (JAXB) and JPA



- Single Common Object Model
- Mapping POJOs
  - JPA Annotations & JAXB Annotations, or
  - JPA XML Mapping File & JAXB Annotations

Common Domain Model



EclipseLink Road Map

- Delivery of monthly incubation milestones
  - Build and testing processes
  - Initial contribution functional
  - 1.0M6 was the last milestone
- 1.0 Release: July 2008
  - Specifications: JPA 1.0, JAXB 2, SDO 2.1
  - OSGi packaging and usage examples
  - Spring Framework support
- Future Enhancements
  - JPA 2.0 – Reference Implementation
  - Database Web Services (DBWS)
  - Data Access Service (DAS)
  - Simplified DataMap Access and Dynamic Persistence

EclipseLink Adoption

- Oracle
  - Oracle TopLink will be a supported distribution of EclipseLink
  - Oracle Application Server's default persistence provider
- GlassFish/SunAS
  - GlassFish v3 using EclipseLink as its persistence provider
- Spring Framework
  - v2.5.2 includes EclipseLink with JPA integration
- Eclipse Ecosystem consumers
  - Dali JPA Tooling Project
  - Teneo to use EclipseLink for EMF model persistence
  - MayInstall for storage of deployment configuration
  - Swordfish Project (SOA) usage of EclipseLink SDO
- Others: Discussions underway

Getting Started with Eclipse JPA

- **EclipseLink:** Eclipse Persistence Services Project
  http://www.eclipse.org/eclipselink
  newsgroup: eclipse.technology.eclipselink

- **Dali JPA Tools**—WTP's tools for JPA development
  http://www.eclipse.org/dali

- **EJB 3.0 & JPA Specifications**
  JPA 1.0: http://www.jcp.org/en/jsr/detail?id=220
  JPA 2.0: http://www.jcp.org/en/jsr/detail?id=317