EclipseWorld 2008

October 28-30, 2008

EclipseLink
Java Persistence Freedom Through XML Binding

Doug Clarke, Oracle

EclipseLink Project co-Lead

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

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.
- Entity Beans/Business Objects
  - 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 number

- JAXP

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

- Using XML binding

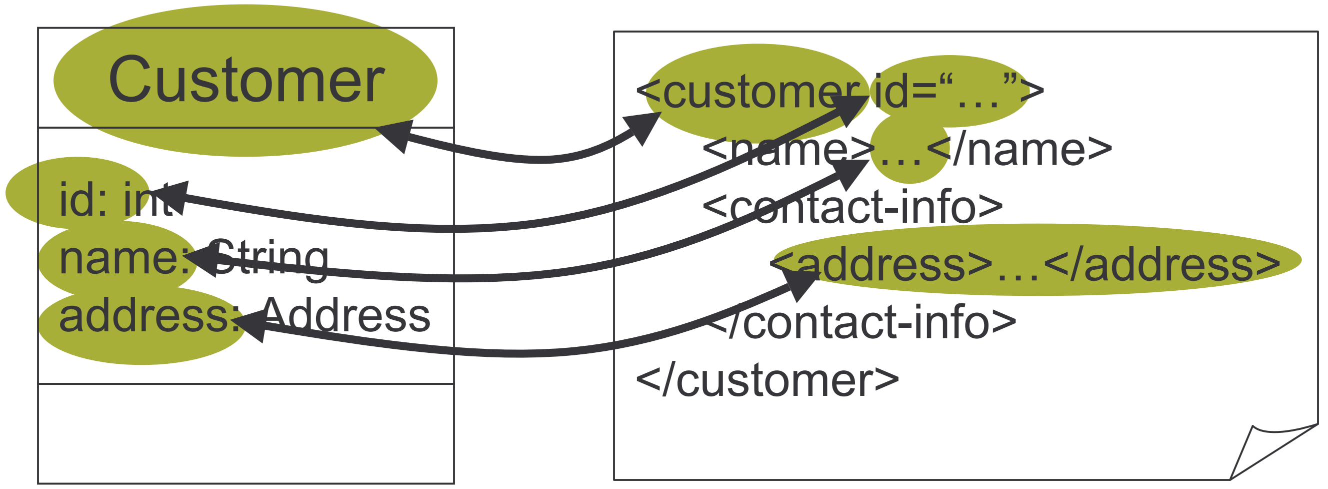
```
employee.getEmployeeNumber();
```

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.

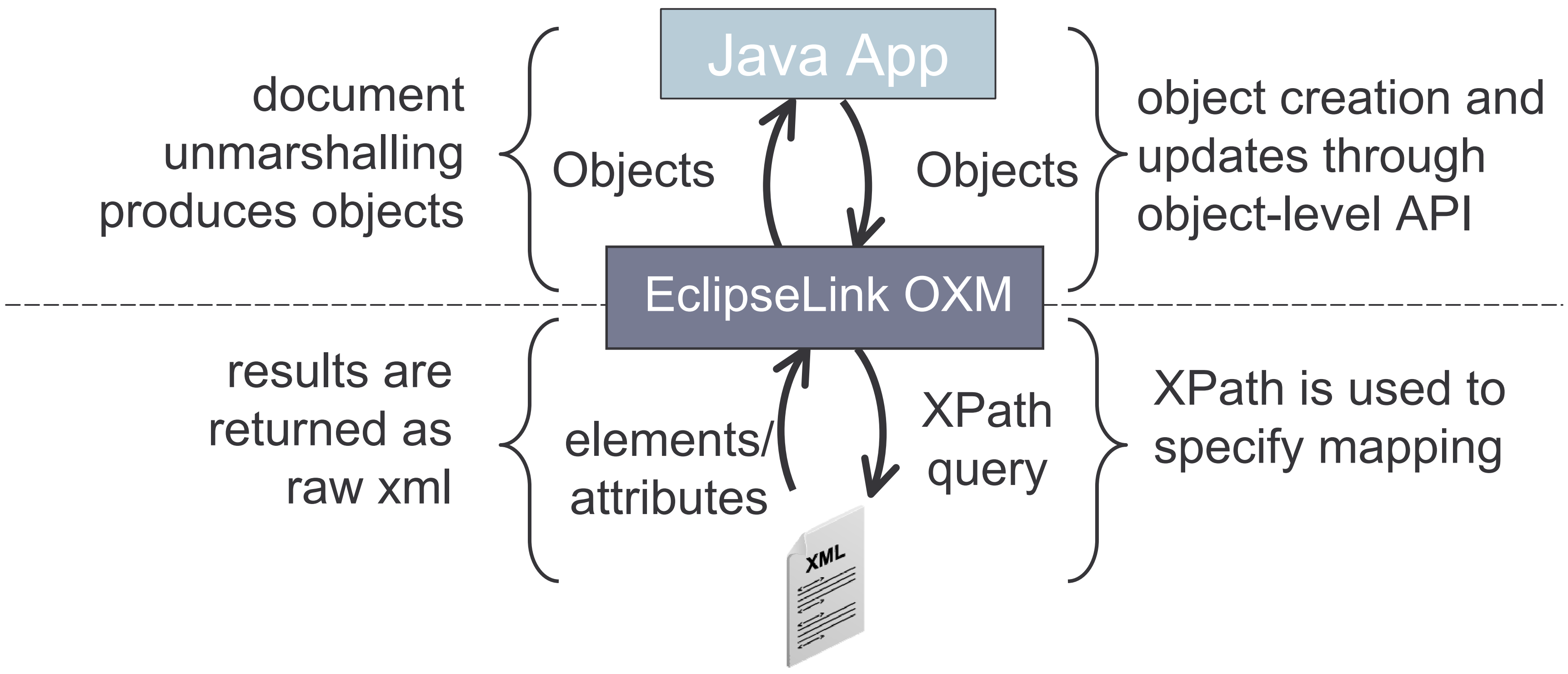


EclipseLink MOXy

“**M**apping **O**bjects to **X**ML”

- Allows developers to work with XML as objects
- Efficiently produce and consume XML
- Provides support for various Object/XML mapping technologies:
  - Java Architecture for XML Binding (JAXB) 2.1
  - Service Data Objects (SDO) 2.1
  - EclipseLink Native OXM / JAXB 1.0

MOXy Binding Layer


document unmarshalling produces objects
Objects
Java App
Objects
object creation and updates through object-level API
EclipseLink OXM
results are returned as raw xml
elements/ attributes
XPath query
XML
XPath is used to specify mapping

EclipseWorld 2008

October 28-30, 2008

JAXB 2

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 environment
  - Java EE Container
  - OSGi
  - Spring

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

JAXB 2—in a Nutshell

- 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

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

MOXy API has Standard API

JAXB 2.0 Standardized Runtime API

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

// 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);
```

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.)

JAXB 2 Object-XML Mappings

- XmlType
- XmlElement
- XmlAttribute
- XmlValue
- And more…

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;
```

Annotations on Properties

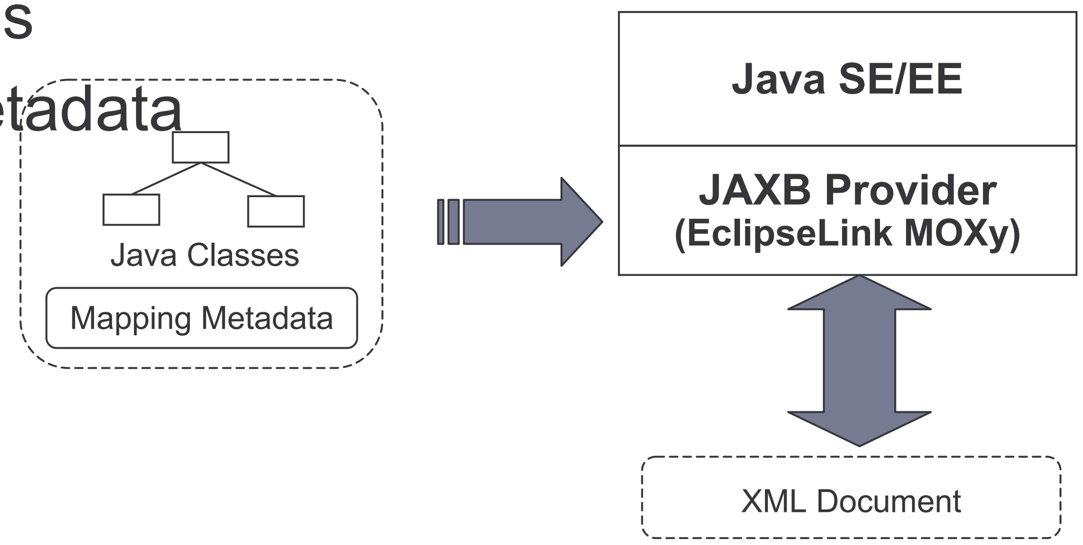
```
@XmlAccessorType(XmlAccessType.PROPERTY)
@XmlType(name = "customer-type", propOrder = {
    "firstName",
    "lastName",
    "billingAddress",
    "shippingAddress",
    "phoneNumber"
})
public class Customer {

    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;
    }
   ...
```

Mappings in XML

- Not defined in JAXB 2
- Expected in next major release.

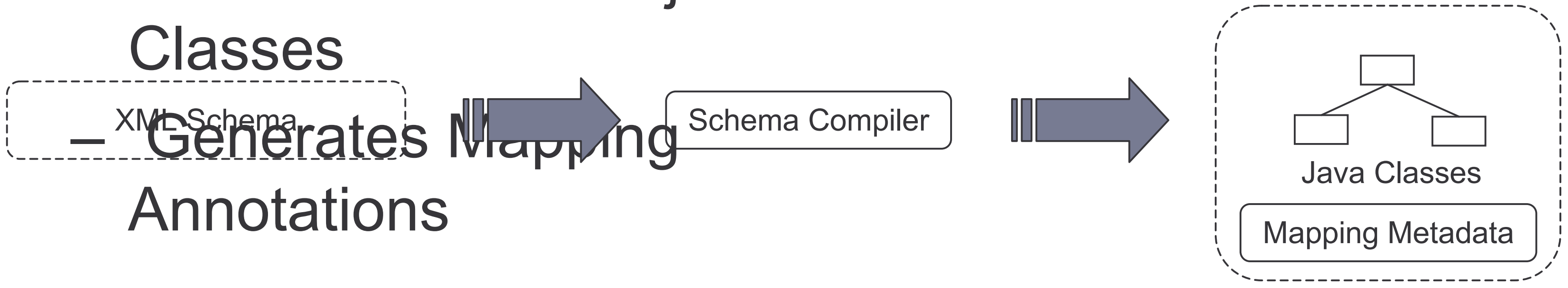
JAXB 2 Runtime

- JAXB runtime combines:
  - Java Classes
  - Mapping Metadata



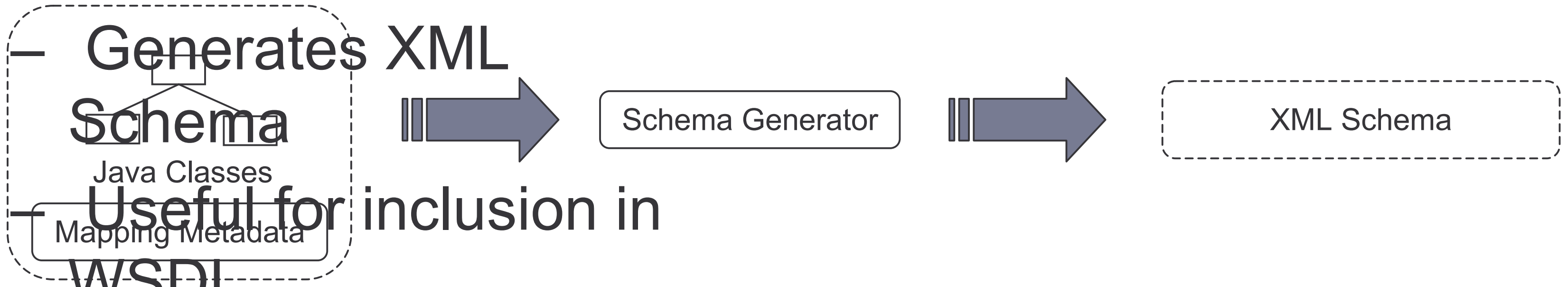
JAXB 2 Design Time—Starting from XML Schema

- JAXB Schema Compiler:
  - XML schema input
  - Generates Java Pojo Classes
  - Generates Mapping Annotations



JAXB 2 Design Time—Starting From Classes

- JAXB Schema Generator:
  - (Annotated) Java Classes input
  - Generates XML Schema
  - Useful for inclusion in WSDL



EclipseWorld 2008

October 28-30, 2008

JAXB 2 Demo

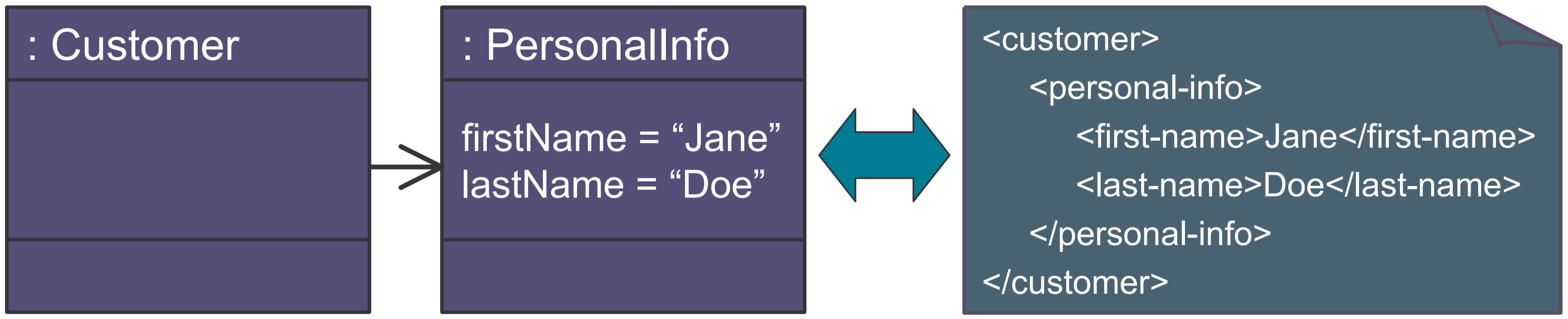
EclipseWorld 2008

October 28-30, 2008

EclipseLink Native OXM/JAXB 1.0

Code Generation

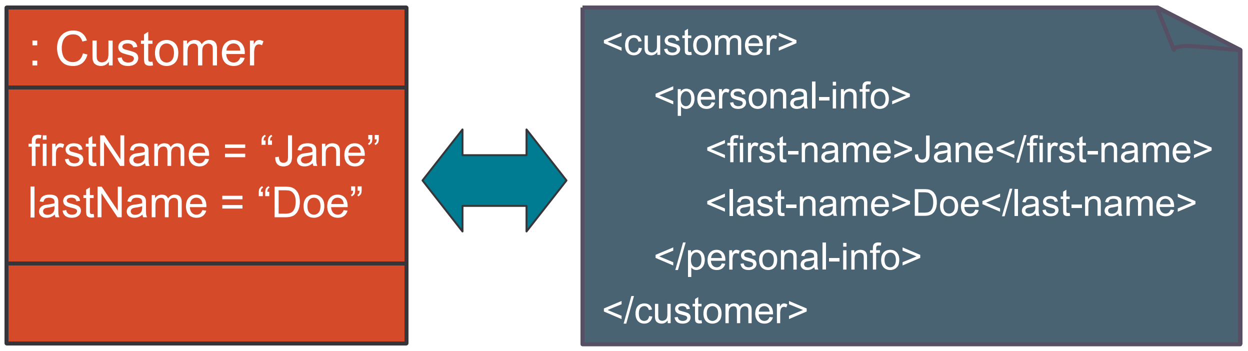
- Sun JAXB 1.0 Reference Implementation
  - Java Classes reflect schema structure
  - Generated classes not extensible/modifiable
  - All document contents marshalled & unmarshalled



```
<customer>
    <personal-info>
        <first-name>Jane</first-name>
        <last-name>Doe</last-name>
    </personal-info>
</customer>
```

Declarative Binding

- MOXy
  - Arbitrary Classes mapped to any schema via mapping metadata



```
<customer>
    <personal-info>
        <first-name>Jane</first-name>
        <last-name>Doe</last-name>
    </personal-info>
</customer>
```

Supported Development Approaches

- Bottom Up: compile schema to generate classes
  - JAXB 1.0 compliant POJOs with external metadata
- Meet in the middle
  - Combine POJOs with external metadata

MOXy's External Mapping Metadata

- Mapping information captured in XML and not in the objects.
- External metadata means this approach is NOT at all intrusive on either the object model or the XML schema.
- The object model can be mapped to <u>multiple</u> XML representations.

Mapping in MOXy

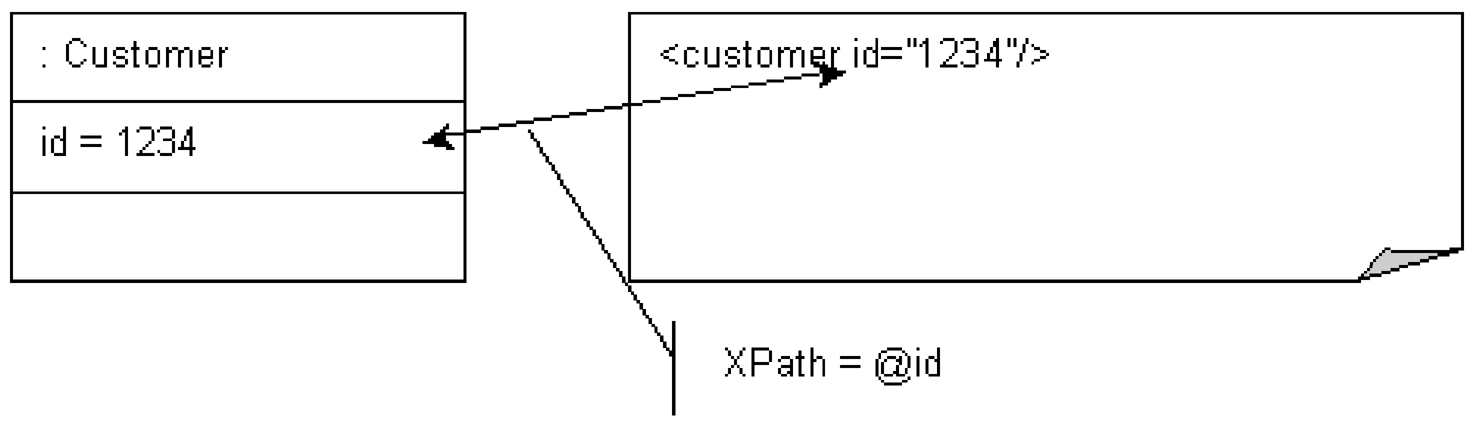
- Powerful mapping approach:
  - XPath based Mapping
  - Positional Mapping
- Extensive Mapping Types
  - Direct
  - Composite Object
  - Composite Collection
  - Direct Collection
  - Relationships
  - Transformation
  - Complex Type Inheritance

XPATH

- MOXy uses XPath expressions to identify XML content that is mapped:
  - XPath by Name
  - XPath by Path and Name
  - XPath by Position
  - Self XPath

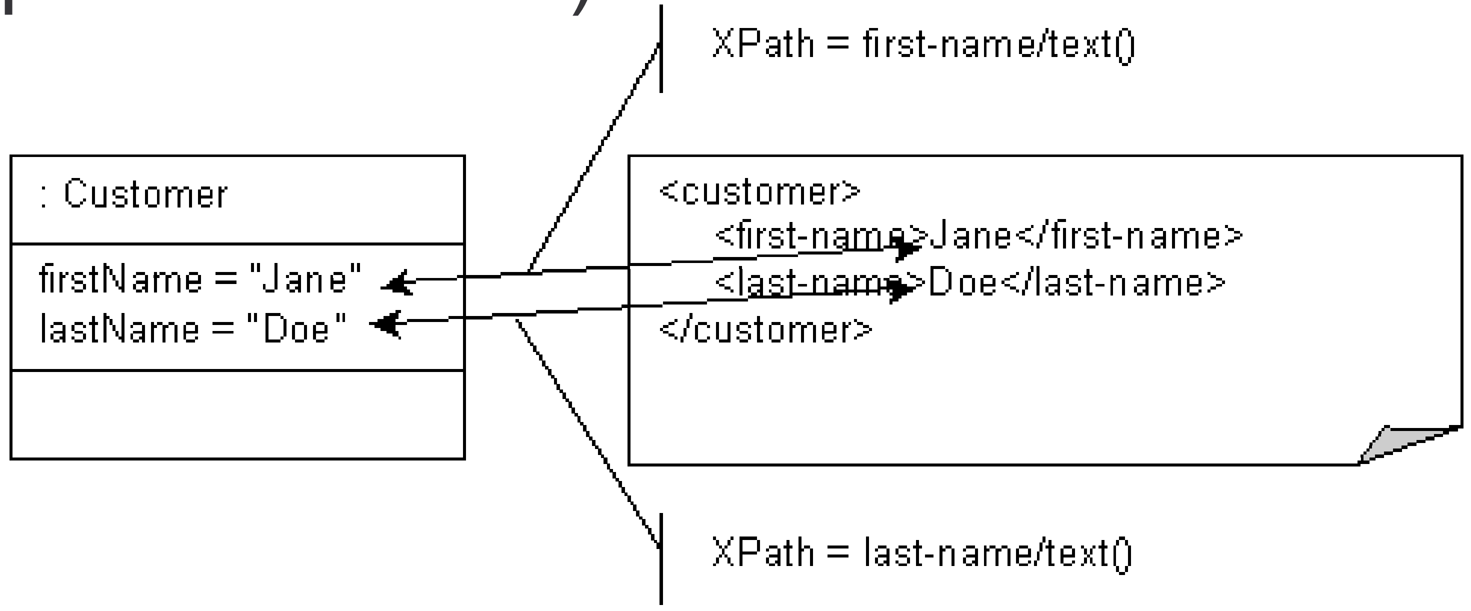
Direct Mapping: Attribute

- Mapping a Java field to an XML *attribute* is done with a DirectMapping and XPath (name).



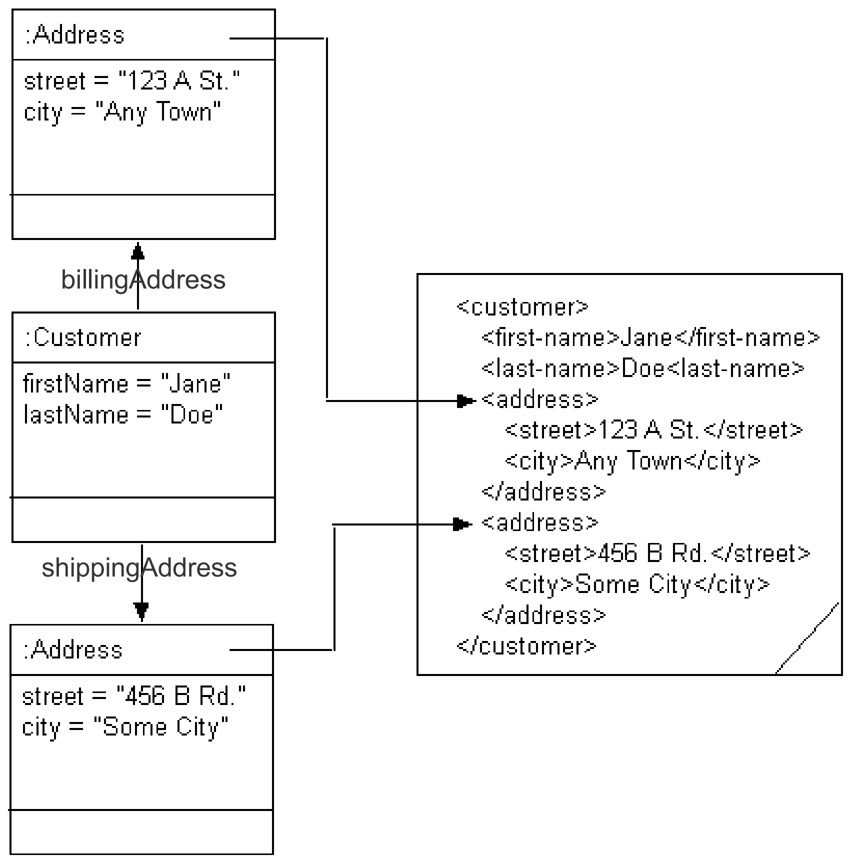
Direct Mapping: Elements

- Mapping a Java field to an XML *element* is done with a DirectMapping and XPath (path and name)



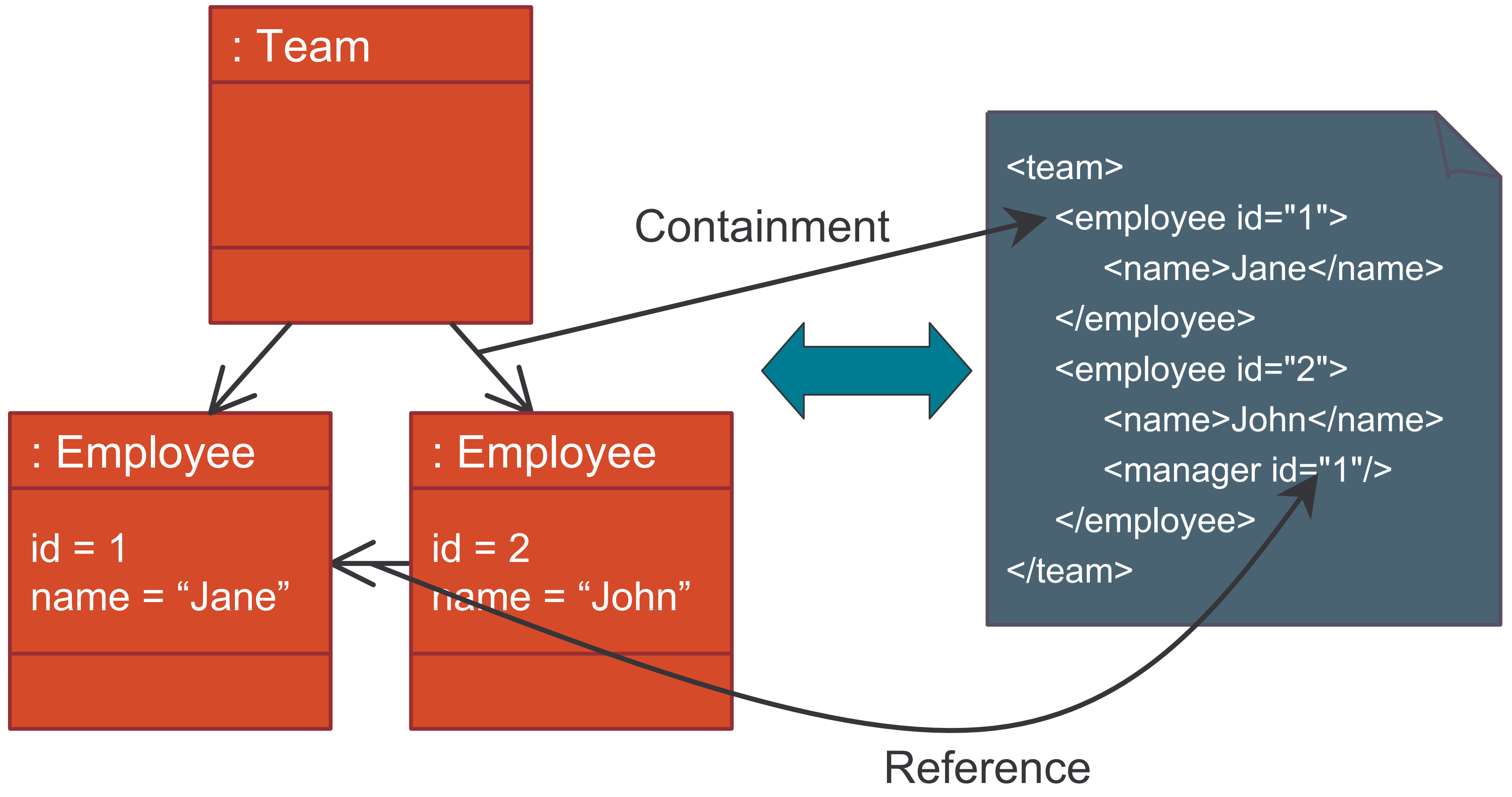
Elements by Position
Example—Composite Object

- An object may have multiple composite object mappings to the same reference class. Each composite object mapping must have a unique XPath, e.g.:
  - billingAddress is address[1]
  - shippingAddress is address[2]



```
<customer>
  <first-name>Jane</first-name>
  <last-name>Doe<last-name>
  <address>
    <street>123 A St.</street>
    <city>Any Town</city>
  </address>
  <address>
    <street>456 B Rd.</street>
    <city>Some City</city>
  </address>
</customer>
```

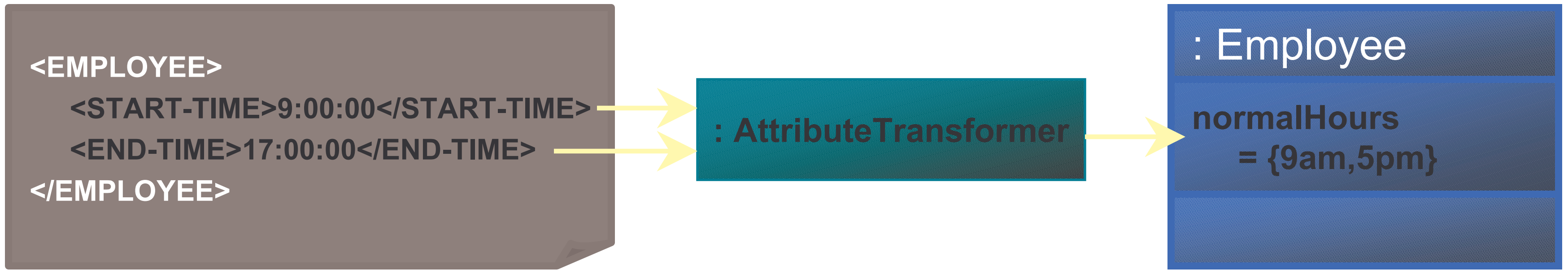
Relationship Support
Containment and Reference (Key-Based)



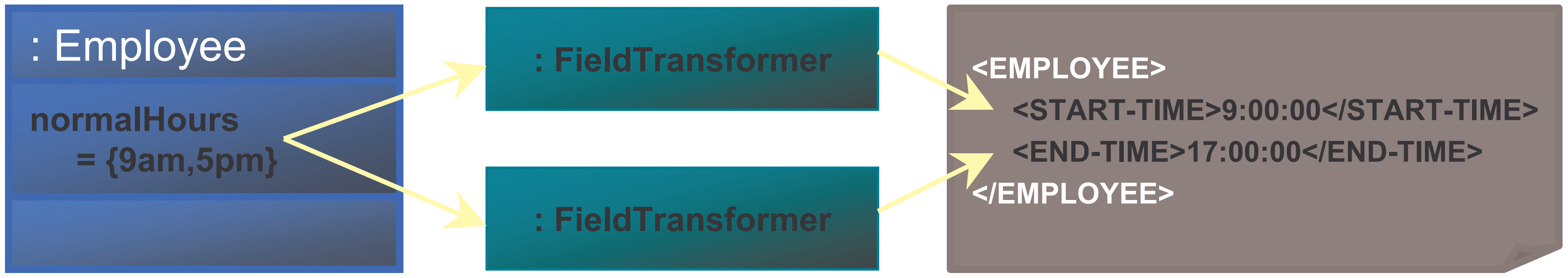
```
<team>
    <employee id="1">
        <name>Jane</name>
    </employee>
    <employee id="2">
        <name>John</name>
        <manager id="1"/>
    </employee>
</team>
```

Transformation Mapping

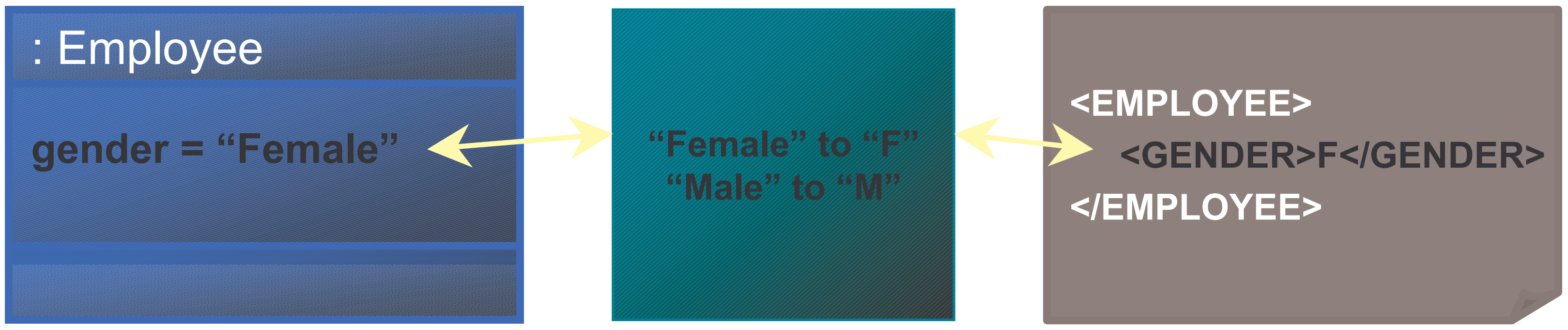
Unmarshal (Read)



Marshal (Write)

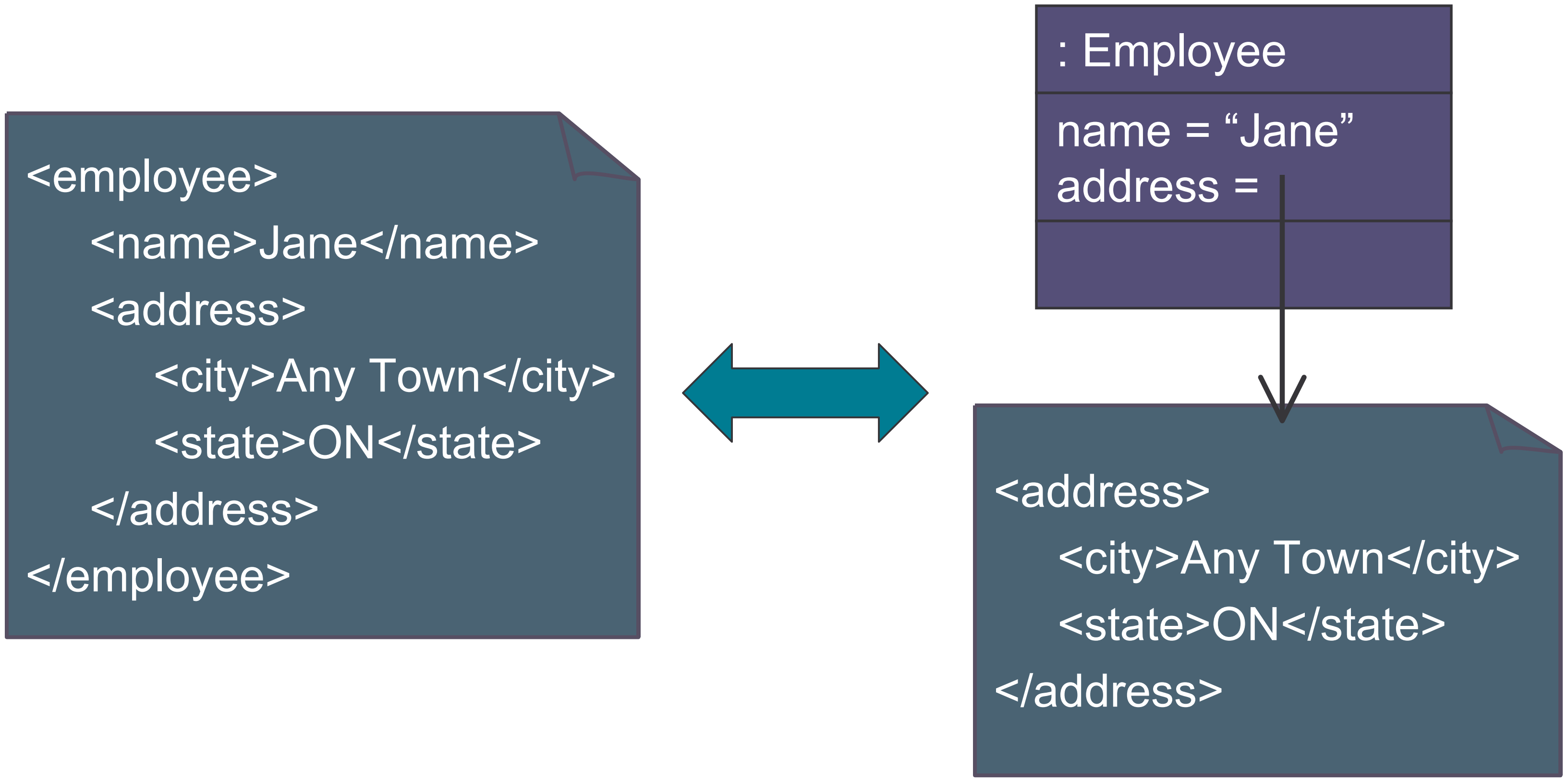


Object Type Converter



Partial XML Mapping

When All Else Fails, Leave it as XML



```
<employee>
    <name>Jane</name>
    <address>
        <city>Any Town</city>
        <state>ON</state>
    </address>
</employee>
```

```
<address>
    <city>Any Town</city>
    <state>ON</state>
</address>
```

MOXy Tooling

- EclipseLink Workbench
  - Part of EclipseLink Utilities component
  - Standalone graphical mapping tool
  - Supports MOXy JAXB 1.0, Native EclipseLink ORM, and EIS
  - Design-time diagnostics
- Java IDE
  - JAXB 2.0 mapping metadata is expressed in annotations.
  - Most IDEs offer syntactic, but not semantic validation.

EclipseWorld 2008

October 28-30, 2008

Service Data Objects (SDO)

What is SDO?

“Service Data Objects (SDO) is a <u>data</u> programming architecture and an API.”

“The main purpose of SDO is to simplify data programming, so that developers can focus on business logic instead of the underlying technology.”

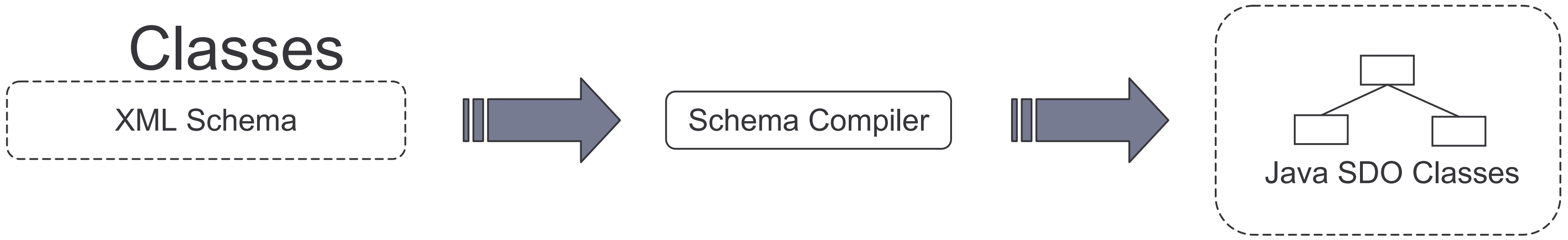
—SDO 2.1 Specification

SDO is XSD-centric

- SDO is for applications centered around XML Schema
- “Static SDO”
  - Classes generated from XSD
  - Classes are not Pojos—they implement SDO Interfaces
- “Dynamic SDO”
  - DataObjects with types/properties derived from XSD

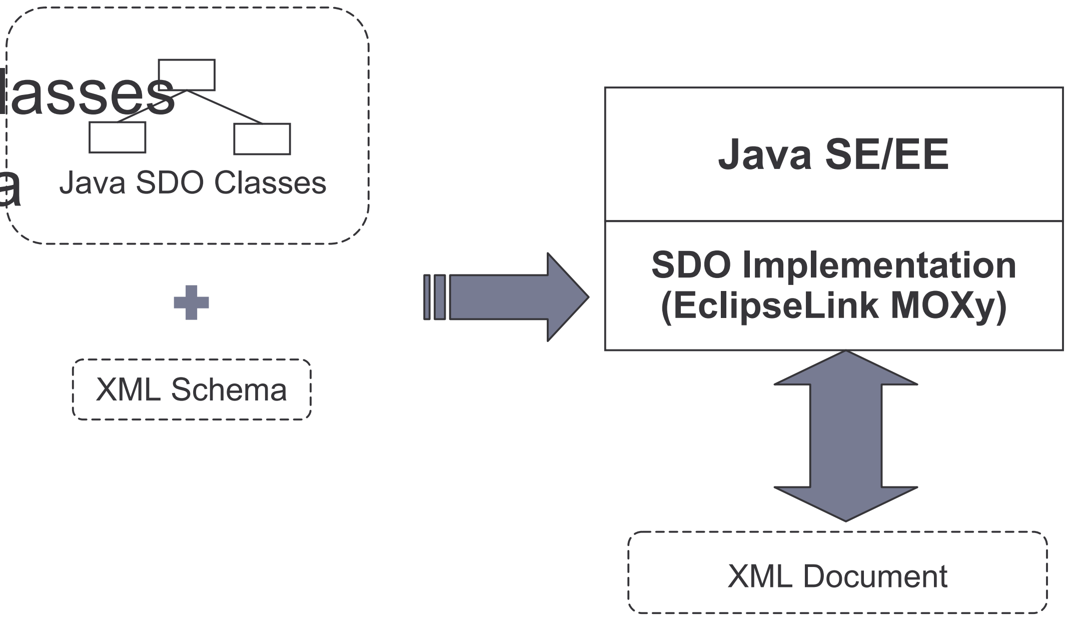
SDO Design Time—Starting from XML Schema

- SDO Schema Compiler:
  - XML schema input
  - Generates SDO Classes



SDO Runtime—“Static SDO”

- SDO runtime combines:
  - Java SDO Classes
  - XML Schema



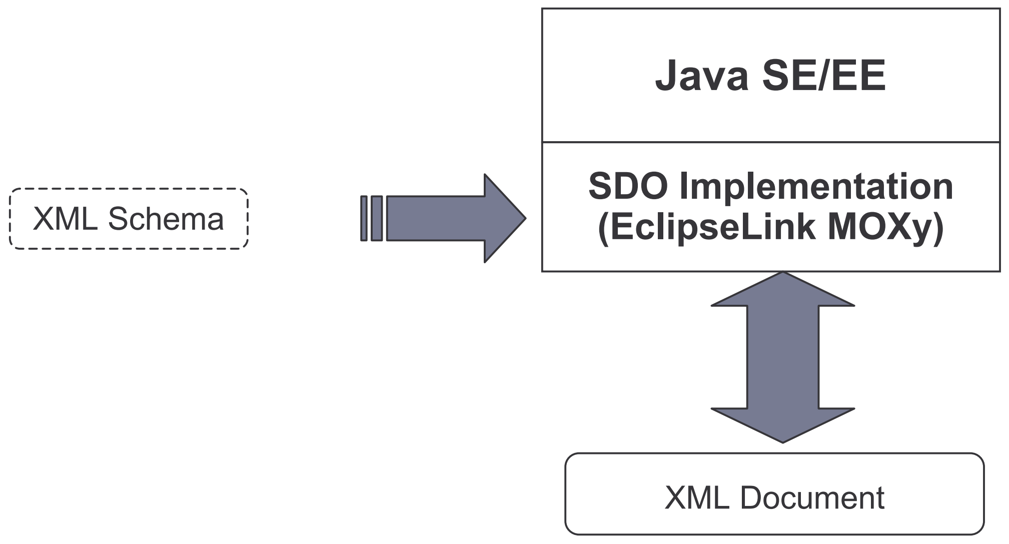
EclipseWorld 2008

October 28-30, 2008

Static SDO Demo

SDO Runtime—“Dynamic SDO”

- SDO runtime using:
  – XML Schema



EclipseWorld 2008

October 28-30, 2008

Dynamic SDO Demo

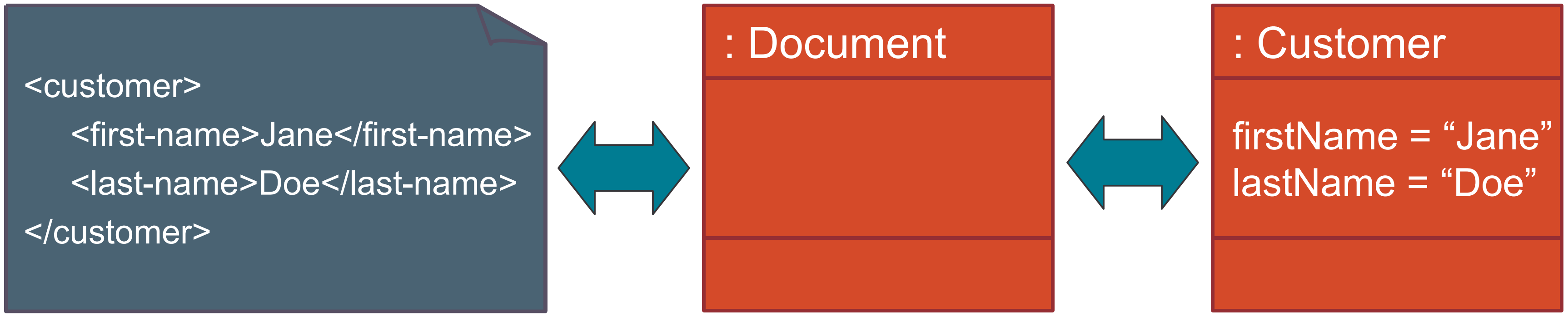
EclipseWorld 2008

October 28-30, 2008

Advantages of MOXy

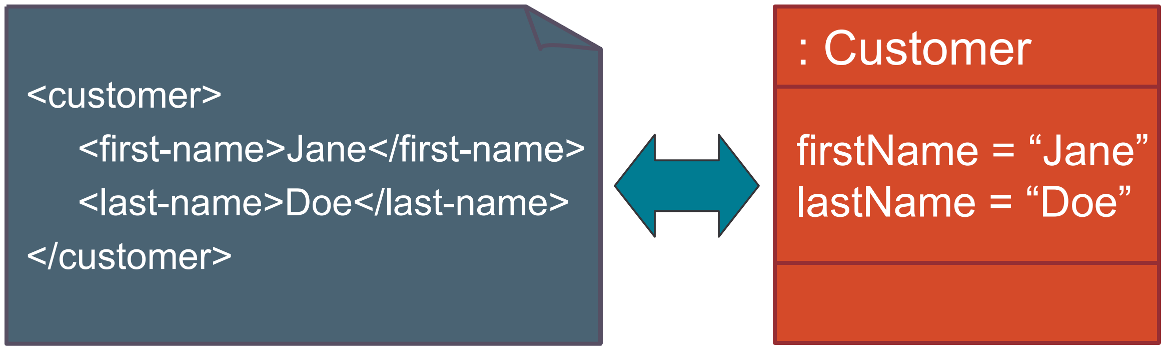
DOM vs. Event Based Parsing

DOM Based – Requires an Intermediate Structure



```
<customer>
    <first-name>Jane</first-name>
    <last-name>Doe</last-name>
</customer>
```

Event Based – No Intermediate Structure Required



```
<customer>
    <first-name>Jane</first-name>
    <last-name>Doe</last-name>
</customer>
```

DOM Based Binding Solutions

Advantages

- Unmapped XML content can be preserved (such as comments).
- User can be given access to the underlying "DOM" structure.

Disadvantages

- Slower and requires more memory
- Underlying "DOM" structure must be built and traversed

Event Based Binding Solutions

Advantages

- Better performance since an intermediate structure need not be built.

Disadvantages

- Unmapped XML content cannot be preserved (such as comments).
- User can be given access to the underlying “DOM” structure

MOXy gives you Choices

- MOXy supports SAX, DOM, and STaX parsers.
- Choose your parsing strategy based on <u>your</u> application needs.

Combining Persistence Services

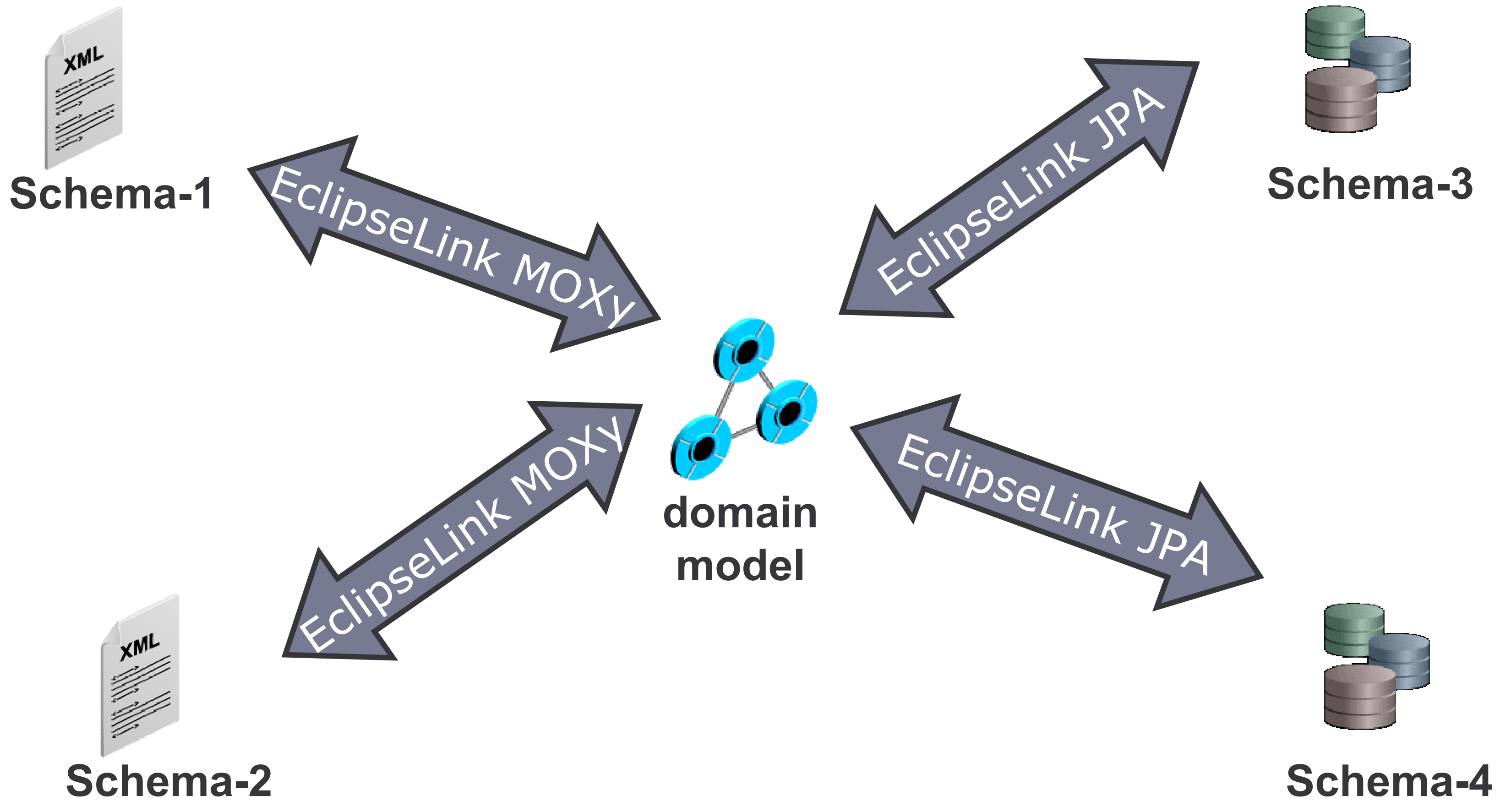
- Metadata based approach allows the same domain model to be mapped with multiple persistence services
  - Supports usage within Web Services/SOA/SCA
  - Domain model can be shared between persistence services (JPA, MOXy, EIS)
  - Transformations are bidirectional:
    - Unmarshall XML to objects and then persist
    - Marshall persistent objects to XML

JAXB 2.0 & JPA 1.0

Combining JAXB 2.0 and JPA 1.0 Annotations

```
@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 address) {…}

}
```

Leveraging Common Domain Model



Spring Integration

- MOXy can be used with Spring through Spring Web Services’ JAXB Marshaller support.
  - Web Services
  - JMS messaging
- For details see: http://onpersistence.blogspot.com/2008/04/eclipselink-moxy-in-spring-ws.html

MOXy Summary

- Usability
- Flexibility
- Performance
- Full W3C XML Schema Support
- Standards Compliance
- Compatibility with Other Standards
- Compatibility with SOA

Q&A