The Evolution of Java Persistence

Doug Clarke
Oracle
Ottawa, Canada

Keywords:

Java, Persistence, JPA, JAXB, JSON, REST

Introduction

The data access requirements of today's Java applications keep expanding and have grown to include features such as tenant data isolation for cloud deployment, extensible models to support per tenant customization, JSON binding for RESTful web services, the ability to store Java objects in NoSQL databases and more. EclipseLink, included in Oracle TopLink, is well known as an object-relational mapping framework and as the JPA 2.0 reference implementation in Java EE 6, but it continues to evolve and now provides a comprehensive set of data services for Java developers building enterprise and cloud applications in Java EE and SE.

Java Persistence: The Problem Space

Java Persistence is most commonly associated with Relational Database usage. Object-Relational mapping was first standardized by the Java Persistence API (JPA) specification within EJB 3.0. However, the scope of the problem space most developers actually deal with in today's application is much larger. It includes not only physical storage of the data but also in transformations dealing with XML and JSON and the service interfaces that require these. The ability to bind objects to XML with JAXB and soon to JSON must also be considered part of the persistence problem space (Figure 1).

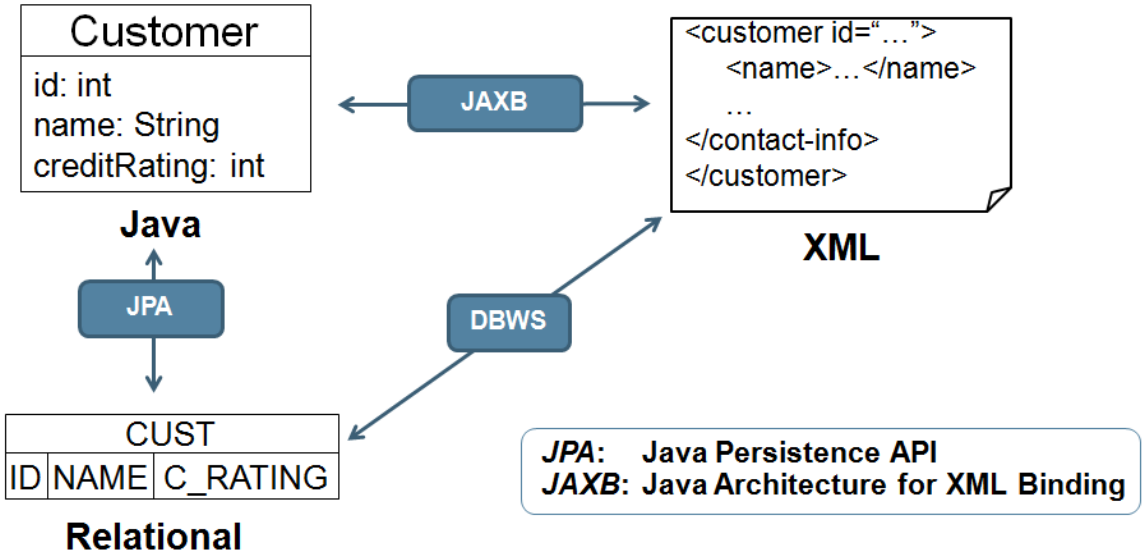


Figure 1: The Problem Space

The ability to leverage persistent data within Web Services, both SOAP (JAX-WS) and REST (JAX-RS) based is of great importance and has lead to the introduction Database Web Service (DBWS) solutions for simplifying how persistent data can be exposed and leveraged.

Java Persistence Landscape

Within the persistence landscape there are several key standards that are available. These include: JPA 2.0, JAXB 2.2, and SDO 2.1.1. Although these are a good base for application development, customers and the technologies they use continue to require innovation within Java Persistence solutions. Some recent innovations we'll look at include:

- JSON Binding
- Dynamic JPA
- Tenant Isolation
- RESTful JPA
- NoSQL

Looking beyond these topics that will be covered in this session there are also new functionality on the horizon in:

- JPA 2.1
- JSON-B
- Standardizing Java-NoSQL

EclipseLink Project

The EclipseLink project (www.eclipse.org/eclipselink) is a key component of the Java Persistence community. EclipseLink delivers the reference implementation for JPA 2.0 and is leading the way with many of the innovations in this space. Within this session we'll highlight some of these innovations as they have been delivered in recent EclipseLink releases.

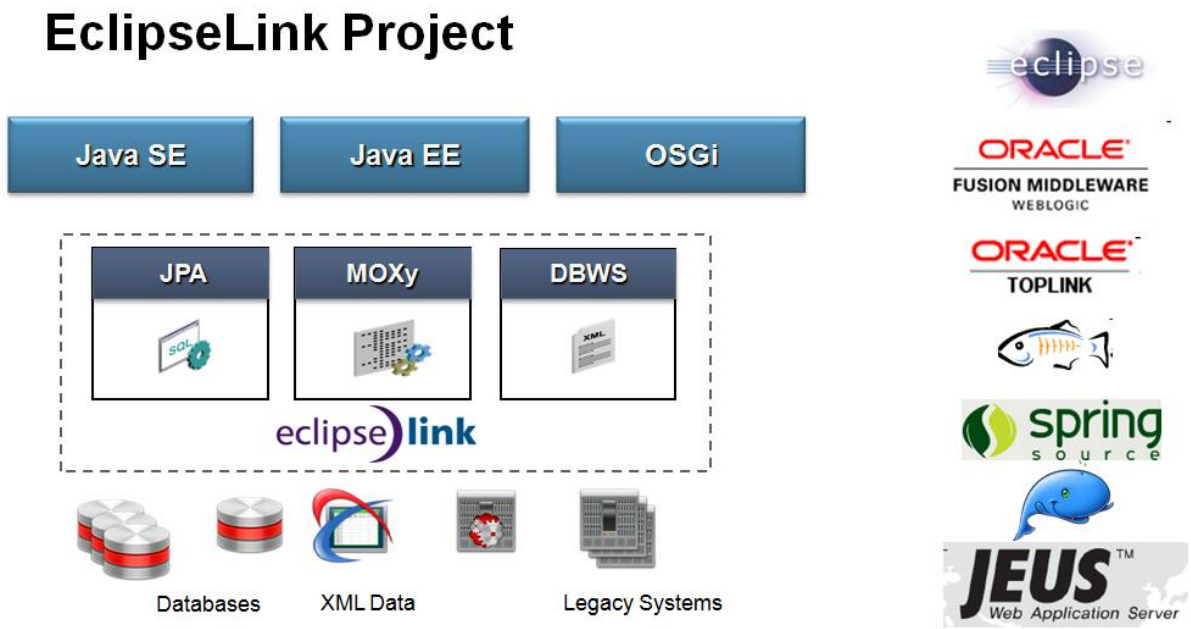


Figure 2: The EclipseLink Project

The EclipseLink Project is composed of three main components:

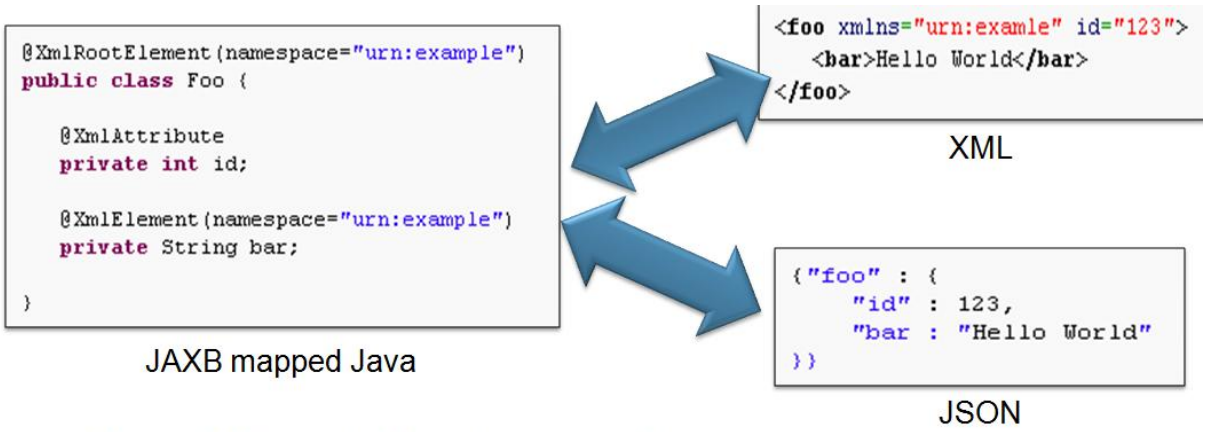
> **JPA**: The object-relational core of EclipseLink enables, optimizes, and scales applications leveraging relational databases. It provides flexible mappings and powerful caching solutions as well as support for extended features of leading databases.
>
> **MOXy**: The JAXB and JSON binding solution simplifies mapping object or JPA entities using standard JAXB annotations or extended XML mapping files. Its fidelity with JPA makes it truly unique and important for developers needing XML or JSON binding support for their JPA entity classes.
>
> **DBWS**: Starting from a JAX-WS based solution focussed on generation of WSDL and supporting infrastructure this component now includes JPA-RS, or RESTful JPA, which simplifies exposing JPA persistence units over REST.

This session builds on these three components explaining how the new features evolved based on customer and technology demands and how they can be easily used within any Java EE or SE application

MOXy's JSON Binding and JPA Fidelity

The EclipseLink project early on recognized the need to map objects into XML based on its own configuration file requirements and the need to support multiple versions of these files to ensure backwards compatibility avoiding the need for customers to upgrade their configuration resources. This support eventually became the Object-XML (OXM) infrastructure and with the creation of the EclipseLink project was named MOXy. This infrastructure evolved to become a compliant JAXB implementation and now serves as the default JAXB implementation within Java EE containers such as Oracle WebLogic.

With the growing demand for JSON support with RESTful services the EclipseLink MOXy team evolved its XML binding solution to also work with JSON. Now developers can easily map their persistent classes to JSON or XML using annotations or XML mappings and can select the format to marshal into at runtime through specification of a media type.




```
Marshaller marshaller = getJAXBContext().createMarshaller();
marshaller.setProperty(MarshallerProperties.MEDIA_TYPE, MediaType.APPLICATION_JSON);
marshaller.setProperty(MarshallerProperties.JSON_INCLUDE_ROOT, false);
marshaller.marshal(entity, writer);
```

In addition to MOXy's expansion into JSON binding it has also adapted to better work with JPA entities. JPA providers generally enhance or extend the JPA entity classes to contain additional state. These enhanced classes and complex mappings do not always work well with JAXB so MOXy has additional support for dealing with these challenges including handling cyclic relationships. This support allows developers to simply map their JPA entity classes to XML/JSON without requiring a parallel object model and the tedious conversion code that goes along with it.

Dynamic JPA

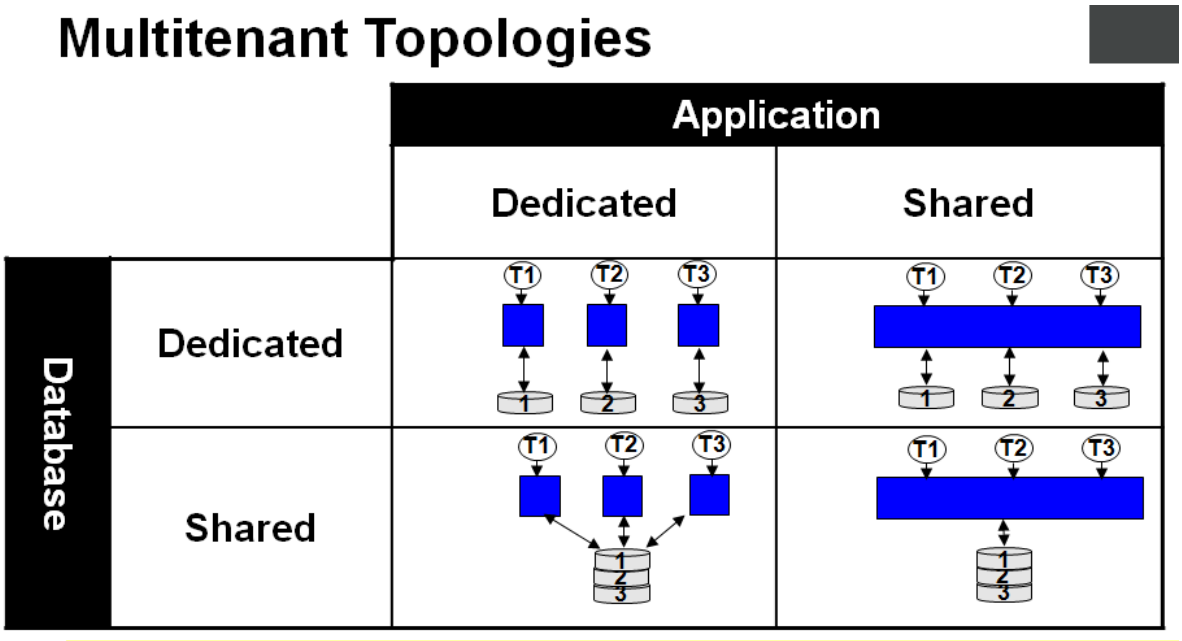
Another area of evolution within EclipseLink is enhanced support for more dynamic JPA models. This dynamic nature of EclipseLink JPA can be seen in several relatively new feature areas:

- JPA Entities without .java or .class: entities defined declaratively in XML mapping files
- Extensible Entities: Entity classes that can have additional mappings defined at runtime

These features were introduced to support application developers who are required to build functionality based on relational structures that were not known at development time. Combined with EclipseLink JPA's meta-model functionality application components can now be developed to leverage future storage requirements.

Tenant Isolation

The ability to support multi-tenant applications where various levels of tenant isolation are required has lead to the introduction of several new features. These features allow EclipseLink based applications to easily share containers and or databases, schema, or even tables. The intent is to simplify how an application can be written and customized or deployed to handle different tenants.



EclipseLink's tenant isolation features for shared database, schema or tables can be easily configured using annotations or XML mapping files.

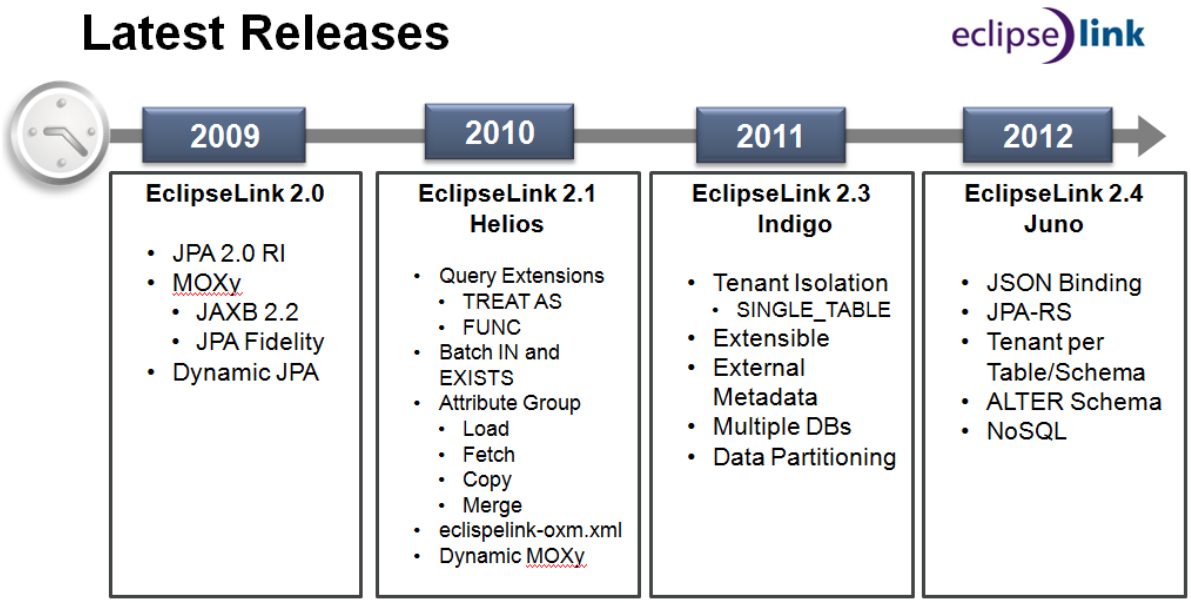
Multitenant Entity Strategies

- GOAL: support storage of entities from multiple tenants in a single shared database/schema/table
- @Multitenant Strategies
  - @Multitenant(SINGLE_TABLE) - default
  - @Multitenant(VPD)
    - SINGLE_TABLE + includeCriteria=false
    - SET_IDENTIFIER(property) & CLEAR_IDENTIFIER
    - DDL Gen of predicate function and ADD_POLICY
  - @Multitenant(TABLE_PER_TENANT)

RESTful JPA

Based on all of the MOXy and JPA extended features, EclipseLink has developed and released its first RESTful JPA solution which delivers a new breed of persistence service. Developers can now expose their existing JPA persistence unit over REST with automated end point definition and XML/JSON binding. Alternatively a dynamic JPA persistence unit can be defined meaning that the entire service definition is declarative and easily be developed, tested, and deployed in a hosted service.

EclipseLink Road Map



Contact address:

Doug Clarke
Oracle Canada ULC
45 O'Connor Street
K1P 1A4, Ottawa
Canada

Email douglas.clarke@oracle.com