UPA Reference Documentation

by Taha Ben Salah

June 9, 2013

Preface

Part I

Introduction

Chapter 1

Présentation

1.1 Why a new Persistence Framework

UPA is mostly like other existing ORM frameworks (Hibernate, MyBatis) and Specifications (JPA) although it follows a slightly new approach : dealing with deeper control on data model : UPA makes finally possible altering database schema in a clean object oriented manner and thus helps providing Untructured Data Model manipulable via structured and unstructured Classes. Although UPA is firstly implemented in the Java Programming Language, it has been designed to be as portable as possible to other object oriented plateforms and progrmaming languages.

1.2 Common Features

1.2.1 Vendor Neural Persistence Layer

UPA helps you build a persistence layer that is vendor neutral and any persistence provider can be used. Although, UPA provides a reference implementation that is particularly performant and ready to use.

1.2.2 Pluggable Providers

UPA supports pluggable, third party persistence providers as it is defined as an API with a reference implementation

1.2.3 Inside/Outside containers

UPA application can run outside the container also. So, developers can use UPA capabilities in desktop applications also.

1.2.4 Annotations based meta-data

No need to write deployment descriptors. Annotations based meta-data very similar of JPA's are supported Besides, annotations defaults can be used in model class, which saves a lot of development time

1.2.5 Standardized ORM

Provides clean, easy, and standardized object-relational mapping

1.2.6 Query language

UPQL is very powerfully query language provided by UPA providing abstraction layer over the persistence model. UPQL makes it possible to avoid specific RDBMS dialects.

1.2.7 Model generation

UPA application can also be configured to generate database schema based on persistence model

1.2.8 Bottom Up / Top Down

[TRANSLATE ME] Fonctionnement en mode Top Down (génération de la base par l'application, recommandé) et Bottom Up (La base de données est supposée existante)

1.2.9 Transaction Management

[TRANSLATE ME] Gestion de transactions avec des niveau de propagation paramétrable (REQUIRED , MANDATORY, SUPPORTS,. . .)

1.2.9.1 Portability

It is also very easy to switch to most performing persistence provider. You can easily move to any commercial persistence providers when needed

1.2.10 Intuitive

UPA is very simple to apprehend for developers that have already worked especially with JPA and JPA annotations.

1.3 UPA Unique Features

1.3.1 Reflexion API

Large applications need better reflexion mechanisms to handle general purpose use cases. This feature enables developers to be aware of used data model at runtime : entities, fields, datatypes, ...

1.3.2 Dynamic data definition and alteration

One unique feature is to enable at runtime data structure alteration by creating new entities or altering existing entities by introducing new fields, removing some fields etc. This feature makes UPA helpful for dynamic model based applications that usually uses vertical tables (columns as rows) which are of very little performance.

1.3.3 Comprehensive model structuring

UPA provides a special model structuring beyond traditional persistence units organized as simple entities of fields. With new concepts of "package", "module", "entity","section" and "field" concepts , software architects can enforce their application modularity by organizing their data model according to their business model and helps creating real application fragments.

1.3.4 Generated & Formula fields

UPA supports natively customizable formula fields. Formula fields are fields for which values are generated according to custom expressions and conditions. Formula field can be used either for generated identifiers or any other field. Generated identifiers may be of any type (not only integers). Other usage of Formula fields include total fields, special incremental fields such as line indices in invoice.

1.3.5 Ready to use Entity Patterns

UPA supports natively Tree Entities for entities referencing them selves with full optimized deep searching. Singleton entities are also supported in a very clean manner mapped as a single row table. Union Entities is another pattern that makes is possible to uniformize search and updates from distinct entities.

1.3.6 Uniform persistence context access

The API can be used in the very same way from Java desktop application, Web containers, EJB containers, Spring container, .Net Destop applications, ASP.Net Web Forms applications, ASP.Net Web MVC applications.

1.3.7 Shared Model

The very same persistence model can be used on multiple database schemas at the same time. Therefore Entity classes are not attached to a specific persistence unit and can be mutualized. Besides, the same classes can be used for mapping different tables in the same schema at the entity definition time. Even better, generic classes can be specified at search time to hold information of some existing entities. One example is to use a generic class holder "NamedEntity" (with solely id and name) to use for all drop down components.

1.3.8 Dynamic Lazy/Partial loading

UPA enables the runtime selection of needed fields/relations for retrieval. this is obviously helpful for performance tuning. In most cases, listing entities does not require retrieval of all information. However entity editing should handle all updatable fields.

1.3.9 Very flexible persistence processing

Entities may be configured to retrieve information from one table, insert to a second table delete using a stored procedure and update using a custom Java class for instance. Rich

1.3.10 Callback System

UPA defines a rich set of Callback mechanisms. Callbacks are available both on Data updates and Data Structure alteration. Data Callbacks enables soft trigger implementations for handling all of inserts, updates and delete operations. Data Structure Callbacks makes it possible to capture Data Structure changes and perform even extra alteration when needed.

1.3.11 Portable support for custom and complex datatypes

UPA supports all common data types, besides it provides a portable manner to extend supported types with new custom/complex ones. This features is similar to Embedded/Embeddable features in JPA although it provides a more extensible manner.

1.3.12 Designed to work equally on Java and .Net platforms

UPA is designed to work with major Object Oriented languages although the very first implementation it provides is with Java(c) and C# (c). Design consideration imposes some restrictions to language advanced features but provides betted concepts understanding, a cleaner implementation and perhaps makes it more intuitive at its usage.

1.3.13 Soft learning curve

UPA provide better performance and a softer learning curve, we have made the choice to make UPQL nearer to SQL than to JPQL. UPA have also native support of views defined using the very same UPQL language.

Chapter 2

Tutorial

create file META-INF/upa.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<upa version="1.0">

    <catalog>

        <schema>

            <connection>

              <connectionString>derby:embedded://testdb/sample</connectionString>
              <userName>upatest</userName>
              <password>upatest</password>
              <structure>create</structure>

            </connection>

        </schema>

    </catalog>

</upa>
```

public class Invoice{
}

Part II

Reference

Chapter 3

Configuration

Chapter 4

API