Working with the STARDAT DDI-Lifecycle Library

6th Annual European DDI User Conference
London, 2014/12/01

Alexander Mühlbauer
GESIS – Leibniz Institute for the Social Sciences
Outline

1 Introduction

2 Architecture Principles

3 Implementation Principles

4 Exercises

5 Wrap-up
Outline

1. Introduction
2. Architecture Principles
3. Implementation Principles
4. Exercises
5. Wrap-up
## Agenda

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>13:30 - 14:00</td>
<td>Welcome</td>
</tr>
<tr>
<td></td>
<td>Workshop objectives</td>
</tr>
<tr>
<td></td>
<td>Introduction of participants</td>
</tr>
<tr>
<td>14:00 - 14:30</td>
<td><strong>Architecture Principles</strong></td>
</tr>
<tr>
<td>14:30 - 15:00</td>
<td>Exercise 1: Getting the demo app run on your local machine</td>
</tr>
<tr>
<td></td>
<td>Questions &amp; Discussion</td>
</tr>
<tr>
<td>15:00 - 15:30</td>
<td>Break</td>
</tr>
<tr>
<td>15:30 - 16:00</td>
<td><strong>Implementation Principles</strong></td>
</tr>
<tr>
<td>16:00 - 16:30</td>
<td>Exercise 2: Reconstruct the historization &amp; versioning show case</td>
</tr>
<tr>
<td></td>
<td>Exercise 3: Create a study description</td>
</tr>
<tr>
<td>16:30 - 17:00</td>
<td>Questions &amp; Discussion</td>
</tr>
</tbody>
</table>
Introduction of Participants

- What is your name? What is your affiliation?
- What is your professional background?
- What do you like most at your daily work right now?
- How high would you rate your experience in DDI C/L? Possible levels: beginner, junior pro, senior pro or guru
- Are you familiar with Java, Spring, Hibernate, Freemarker, Maven, Git, Eclipse?
- What are your expectations of this tutorial?
Tutorial Objectives

After this workshop, you will be able to

- download, compile and (re)use all source and binary code of the library and demo application
- understand the basic design principles of the library
- create simple study descriptions in the formats DDI 3.1 and DDI 3.2
STARDAT Project at a Glance

- Integrated management system for standardized metadata documentation
- Replacement for production systems DBK, DSDM, CBE of GESIS data archive
- Support of and high interoperability with DDI-C and DDI-L
- Longterm-preservation with DDI
Classification of the Library

- Open source,
- Java-based,
- extensible
- domain model library
- with object-relational persistence
- to support DDI Lifecycle metadata documentation

- Hybrid between *Contract First* and *Code First!*
Distinction of Design Principles

- Architecture principles
  - Abstract, high level concepts
  - Technology independent
  - All-or-Nothing decisions

- Implementation principles
  - Concrete, low level realisations
  - Technology dependent
  - Balancing priorities

Feedback appreciated!

- Obviously, classification is ambiguous and criticisable!
Outline

1. Introduction
2. Architecture Principles
3. Implementation Principles
4. Exercises
5. Wrap-up
Separation of Model and Representation
Contract First – XML Schema-based Approach

User Interface DDI 3.1
  ↓
Services DDI 3.1
  ↓
Model DDI 3.1
  ↓
Representation DDI 3.1

User Interface DDI 3.2
  ↓
Services DDI 3.2
  ↓
Model DDI 3.2
  ↓
Representation DDI 3.2

User Interface DDI 3.X
  ↓
Services DDI 3.X
  ↓
Model DDI 3.X
  ↓
Representation DDI 3.X

...
Separation of Model and Representation
Hybrid between *Contract First* and *Code First*!
Separation of Model and Representation
Architecture Principle: SMORE

- leads to true abstraction in memory
- is the very necessary base for true interoperability
- Example for mixing: Coverage
Separation of Model and Representation

ddi:Citation
Separation of Modeling and Mapping
Architecture Principle: SMOMA

- avoids specific, not reusable implementations
- allows use case specific, possibly different mappings
- makes incompatibilities explicit
- Examples for mixing
  - dcterms in ddi:Citation
  - Everything as ddi:Note
  - Creating a type PrimaryResearcher instead of explicitly mapping information on ddi:Creator, ddi:Publisher or ...
Extensible object-oriented model

Hierarchical Approach

- Specification Implementation
  - Extension and Adaption on Organizational Level
    - Extension and Adaption on Project Level

Compositional Approach

- Specification Model Class
  - Organizational Model Class
  - Project Model Class
    - Specification Model Class
      - Project Model Class
        - Organizational Model Class
Extensible object-oriented model
Comparison of the Two Approaches

- **Hierarchy (Inheritance)**
  - Technical infrastructure on organizational level (+)
  - Support of services out of the box (+)
  - Lower-level components must be true extensions (-)
  - Explicit definition of requirements for extensibility (-)

- **Composite (Mapping)**
  - Independence and flexibility in modeling (+)
  - Making incompatibilities explicit (+)
  - Re-implementation of components (-)
Extensible object-oriented model
Implementation Challenge: Loss of Maintainables
Separation of *Model* and *Validation*

Architecture Principle: SMOVA

- Validation requirements may differ relating to agencies, projects, time and DDI versions
- Usage of validation strategies for every resource according to requirements
- Domain model objects are only data transfer objects (DTO).
- Model invalidating constraints are rules of logic.
- Model is structure, validation is process.
Container-managed transactions
Architecture Principle: CMT

- Transaction management delegated to IoC container
- Services as (a)synchronously executable units of work
Exercise 1
Getting the demo app run on your local machine

- [ ] http://stardat.codenomics.de/historization-and-versioning/getting-started
- [ ] run it locally with `mvn clean tomcat7:run`
Outline

1. Introduction
2. Architecture Principles
3. Implementation Principles
4. Exercises
5. Wrap-up
Objects either of Type Resource or Property
Implementation Principle: ORP

A resource may have a resource as its context. The context owns its associated resources.

A resource owns its associated properties.

A resource may reference other resources.
Objects either of Type Resource or Property
Implementation Principle: ORP

- A property must be associated with only one, owning resource.
- Any object which has Uniform Resource Names is of type Resource.
- Any object which is not of type Resource is of type Property.
- From persistence view Resource is an @Entity, Property is a @MappedSuperClass.
Domain Semantic Objects of Type Resource

Implementation Principle: OSR

- Distinction of identifiable, versionable, maintainable objects not reasonable
  - ddi31:r:Abstract as identifiable, but never referencable?

- A resource is
  - a domain object
  - which is maintained by an agency
  - with identity
  - across the whole lifecycle.
Java Class Resource
with JPA and Hibernate Envers Annotations

```java
@Audited
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class Resource implements ChangePropagatable {
  @Id
  @GeneratedValue(strategy = GenerationType.TABLE)
  private Long primaryKey;

  @Column(columnDefinition = "BINARY(16)")
  private UUID hashCodeSurrogate;

  @Column
  private long timestamp;

  public abstract Resource getContext();

  // ...

  @Override
  @PrePersist
  public void propagateChange() {
    timestamp = RevisionContext.getInstance().getTimestamp();
    if (getContext() != null)  
      getContext().propagateChange();
  }
```
Separation of Int. and Ext. Identification

Reference by Uniform Resource Name
Reference by Primary Key and Revision
Separation of Int. and Ext. Identification

- **Internal identification**
  - Meet technical requirements of persistence technology by primary keys
  - Efficient storage and querying by single-valued, flat identifiers without any semantics

- **External identification**
  - Meet business requirements of domain by uniform resource names
  - Distributed resolution of multi-valued, hierarchical identifiers with semantics
Separation of Historization and Versioning
Definition: **Historization**

- is the technical process
- to keep track of change
  - which metadata objects changed
  - when
  - why
  - by whom
- within a transaction
- on attribute and relationship level;
- builds the foundation of **Versioning**.
Definition: **Versioning**

- is the business process
- to flag a given metadata object
- at a given revision with a version number
- according to the agency’s versioning policy
- if possible with recommendation or automatically.

**Note!**

- Revision number is a global identifier of repository state.
- Version number is not a property of a metadata class.
Outline

1. Introduction
2. Architecture Principles
3. Implementation Principles
4. Exercises
5. Wrap-up
Exercise 1
Getting the demo app run on your local machine

- [http://stardat.codenomics.de/historization-and-versioning/getting-started](http://stardat.codenomics.de/historization-and-versioning/getting-started)
- run it locally with `mvn clean tomcat7:run`
Exercise 2
Reconstruct the historization & versioning show case
Outline

1 Introduction

2 Architecture Principles

3 Implementation Principles

4 Exercises

5 Wrap-up
Wrap-up

- Download, compilation and (re)use of all source and binary code of the library and demo application
- Introduction into the basic design principles
- First steps to create simple study descriptions in the formats DDI 3.1 and DDI 3.2
Thank you!
Enjoy upcoming EDDI!

alexander.muehlbauer@gesis.org