Lessons-learned from Using DDI-RDF Discovery Vocabulary as Backend Model

EDDI14 – 6th Annual European DDI User Conference

Matthäus Zloch
matthaeus.zloch@gesis.org
PhD Student, M.Sc. CS
GESIS - Leibniz Institute for the Social Sciences
Levels of Lessons-learned
Levels of Lessons-learned

Abstract Model → Model Implementation → Abstract Persistence → Persistence Implementations

Levels of Lessons-learned

• Levels
• Views on data
• Statistics
Levels of Lessons-learned

Abstract Model → Model Implementation → Abstract Persistence → Persistence Implementations
About Modeling in General

• Conceptual data model is developed according to a requirements document
• Good practice: use abstract model and extend it to own needs
About Persisting the Model

• The model shouldn’t be restricted to a physical persistence type
• Persistence types exchangeable by configuration
• Data model must not be driven by the views of the application on your data
Why we have chosen DDI-RDF

Requirements

• General enough as “native” application model
• Separate model from application
  – In order to be used in other projects
• Export easy
  – No mapping to the standard schema
  – Iterate through object-structure
DDI-RDF and JPA

• Implemented abstract model as Java classes
• Annotated with JPA
  – Persistence model for object-relational mappings
  – Can be used with any implementation of JPA
  – Creates entity types on physical layer
  – Should be a matter of configuration
• Focus on API design and code reuse

@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class Resource extends PersistableResource {
  @Column
  private String versionInfo;
  @OneToOne
  private LangString prefLabel;
}

@MappedSuperclass
public abstract class Union_StudyGroupStudy extends Resource {
  @ManyToOne
  private Concept kindOfData;
  @ManyToOne
  private List<AnalysisUnit> analysisUnit;
  @ManyToOne
  private List<Universe> universe;
}

@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class Study extends Union_StudyGroupStudy {
// relations
  @ManyToOne
  private StudyGroup inGroup;
  @ManyToMany
  private List<Variable> variable;
  @ManyToMany
  private List<LogicalDataSet> product;
}

@Entity(name = "Missy_Study")
public class Study extends org.gesis.discovery.Study {
  @Lob
  private String note;
}

@Entity(name = "Missy_Variable")
public class Variable extends org.gesis.discovery.Variable {
  @Column
  private boolean derived = false;
  @OneToMany
  protected List<Concept> concept;
  @OneToMany
  protected List<Question> question;
}
Levels of Lessons-learned

- Abstract Model
- Model Implementation
- Abstract Persistence
- Persistence Implementations
Persistence Level

- Architecture you want to access your data through
- Good practice to abstract *method access* away from *how you access data*
- Business Level does not need to know *how* the data is stored
Levels of Lessons-learned

Abstract Model

Model Implementation

Abstract Persistence

Persistence Implementations
DDI-RDF and JPA

• JPA annotations
  – Can be used with any implementation of JPA to materialize the model
The diagram illustrates the structure and dependencies of various components within a software system. It includes:

- **Layers**: Data Storage Access
- **Maven Modules**:
  - `missy-persistence-api` connected to `org.gesis.ddl::disco-persistence-api`
  - `missy-persistence-relational` connected to `disco-persistence-relational`
  - `missy-persistence-xml` connected to `disco-persistence-xml`
  - `missy-persistence-rdf` connected to `disco-persistence-rdf`

The diagram uses arrows to indicate dependencies and connections between these components. It highlights the use of database technologies such as PostgreSQL (RDB), XML, and Virtuoso Triple Store, showing how they interact with the various parts of the system.
Storage Types

• Implementation of model classes is highly hierarchical

• How does the storage type save the data?
Storage Types

• Relational Database – Tables
  – Use InheritanceType.JOINED
  – Use InheritanceType.SINGLE_TABLE

• Graph Database – Nodes

• RDFStore – Triples
Example: Relational Database

• One table for each @Entity
  – Clean database
  – Many (unnecessary) tables involved in query
  – Updates affect several tables

FROM

missy_variable, variable, concept, resource,
missy_logicalDataSet, logicalDataSet, resource, logicalDataSet_variable
missy_study, study, resource, study_logicalDataStaet
Levels of Lessons-learned

- Levels
- Views on data
- Statistics
Variable name: DB040
Variable label: Region (NUTS 1 or 2)
Classification: NUTS
Reference Period: constant
Description Target Variable: Refers to the region of the residence of the household at the date of interview.

Country Specific Comments:

Other Comments:

Thematic Classification: Region (NUTS 1 or 2)
Filter: household

Is ad-hoc module variable: No
Is derived variable type: No

Question Text:

Question Wording: technical item

Comment:
Views on data

• Specific parts of a model
  – Project specific
  – Use case specific

• Good practice: create views on the physical level with the introduction of new entities
Levels of Lessons-learned

• Levels
• Views on data
• Statistics
Functional Statistics

- StudyGroup 5
- Study 50
- LogicalDataSet 90
- Countries 33

- CategoryStatistics 1.3M
- SummaryStatistics 290 TSD
- Variable 12 TSD
- Document 5 TSD
- Question 2,7 TSD
Technical Statistics

- Intel XEON 2.6Mhz, 2GB RAM, 40GB HDD
- MySQL default installation on Debian 6
- 1GB HDD space usage by MySQL
- 150 Tables
Conclusions DDI-RDF

- DDI-RDF after standardisation, is ready to be implemented
  - As back-end model in different projects
  - With different persistence types
- Open Source frameworks provide many ways to get your data persisted
- It is possible to generate a framework for disco that may be extended
Conclusions Code-Reusage

• Do not create isolated, project specific software
• Create (software) pieces that are reusable
• Reuse other software pieces and/or customize it to your own needs
Contribute and Share

• Go to GitHub
• Download missy-project / disco-model-impl
• Discuss and Contribute

Matthäus Zloch
matthaeus.zloch@gesis.org
PhD Student, M.Sc. CS
GESIS - Leibniz Institute for the Social Sciences