gesis
Leibniz Institute for the Social Sciences

Creating a Prototype Application Compatible with DDI 3.1 for the STARDAT Project

Alexander Mühlbauer

3rd Annual European DDI Users Group Meeting
DDI - The Basis of Managing the Data Life Cycle
December 5 - 6, 2011, Gothenburg, Sweden

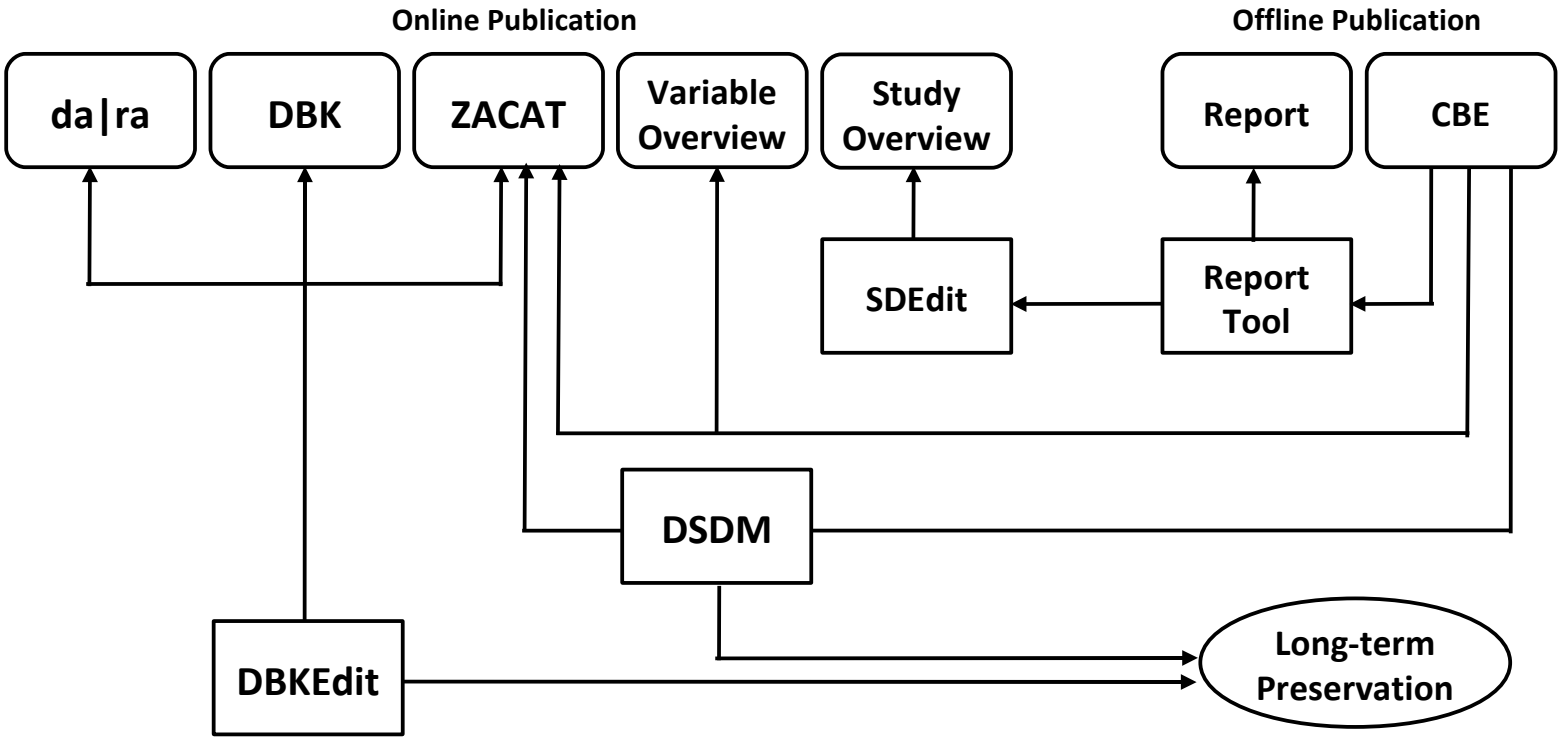
Overview

- Initial Situation and Intention of the STARDAT Project
  - Different Archiving Tools
  - Integration of Different Archiving Tools
  - DDI Formats Currently Used

- Basic Architectural Foundations derived from Prototyping
  - Mapping our Grown Data Structures to DDI 3.1
  - DDI 3.1 Class Modeling with Object-Relational-Mapping
  - Communication between Clients and Server
  - Concept of Historization and DDI Versioning
  - Undo Mechanism During Documentation Process

Initial Situation and Intention of the STARDAT Project

Different Archiving Tools



DBK	Data Catalogue
ZACAT	Online Study Catalogue
DBKEdit	Data Catalogue Edit-Tool
SDEdit	Editing-Tool for Study Method Reports
DSDM	Dataset Documentation Manager
CBE	CodebookExplorer
da|ra	Registration Agency

Initial Situation and Intention of the STARDAT Project

Different Archiving Tools

- Data Catalogue
  http://www.gesis.org/en/services/research/english-question-text/

- Online Study Catalogue
  http://zacat.gesis.org/webview/

Initial Situation and Intention of the STARDAT Project

Integration of Different Archiving Tools 1

- Integrated management system for metadata
- Transfer of the features of DBKEdit, DSDM, CBE and further tools
- Interoperability with standards like DDI-C, DDI-L and ISO 20252
- Multi-language documentation on study and variable level
- Web based modul for structured metadata capture, management and dissemination (Web Based Data Ingest)

Initial Situation and Intention of the STARDAT Project

Integration of Different Archiving Tools 2

- Controlled vocabularys (Thesauri)
- Related publications, continuity guides, scales, trends and additional metadata
- Longterm-preservation with DDI
- Export in different portals like ZACAT, Cessda Data Portal, Sowiport

Initial Situation and Intention of the STARDAT Project

DDI Formats Currently Used

- Export to DDI 2.0 and DDI 2.1
  - for publication on ZACAT (Nesstar) server
  - for data exchange with portals like da|ra and sowiport
  - for long-term archiving
- Export to DDI 3.1
  - for Enhanced Publication editor (linking publications to datasets)

Initial Situation and Intention of the STARDAT Project

Requirements Concerning DDI Formats 1

- Export to DDI 2.1 still needed
  - for publication on ZACAT (Nesstar) server
  - for data exchange with portals like da|ra and sowiport
- Export to DDI 2.5 needed
  - for upgrading metadata to DDI 3
- Export to DDI 3.1 needed
  - for long-term archiving
  - for Enhanced Publication Editor (linking publications to datasets)
- Import from DDI all versions needed
  - for data exchange with primary researchers/projects

Initial Situation and Intention of the STARDAT Project

Requirements Concerning DDI Formats 2

- Future DDI versions support needed
- Usage of rescource packages for reusing elements needed
  - for elements of our own and other institutions
- Concept for long-term archiving of reused elements needed
  - for long-term archiving
  - for Enhanced Publication Editor (linking publications to datasets)

Basic Architectural Foundations from Prototyping

Mapping our Grown Data Structures to DDI 3.1 Really Internalize Lifecycle Orientation

- Managing documentation process of complex social science data
  - Apply adequate grouping approach
  - Identify a strategy to establish resource packages

- Migration issues
  - Find equivalent elements
  - Identify additional elements needed
  - Identify reusable elements
  - Handle with not mappable types

- Building software
  - Existing software tools are *static*
    → Only their combination “supports” lifecycle management
  - New software tool shall be *dynamic*
    → Lifecycle management is inherently contained

Basic Architectural Foundations from Prototyping

DDI 3.1 Class Modeling with Object-Relational-Mapping What Does It Mean When We Talk About DDI 3 Usage?

- Supporting DDI 3
  - Proprietary domain model
  - Proprietary storage
  - → I/O module with some squeezing mapping
- Compatible with DDI 3
  - DDI 3 domain model, perhaps some proprietary extensions
  - Storage in relational database
  - → mapping between XML and relational database
- Based on DDI 3
  - DDI 3 domain model, no proprietary extensions
  - Storage in flat XML files or native XML databases
  - → no mapping, full first-level interoperability

Basic Architectural Foundations from Prototyping

DDI 3.1 Class Modeling with Object-Relational-Mapping Hiearchical Nested vs. Flat Relational Structures

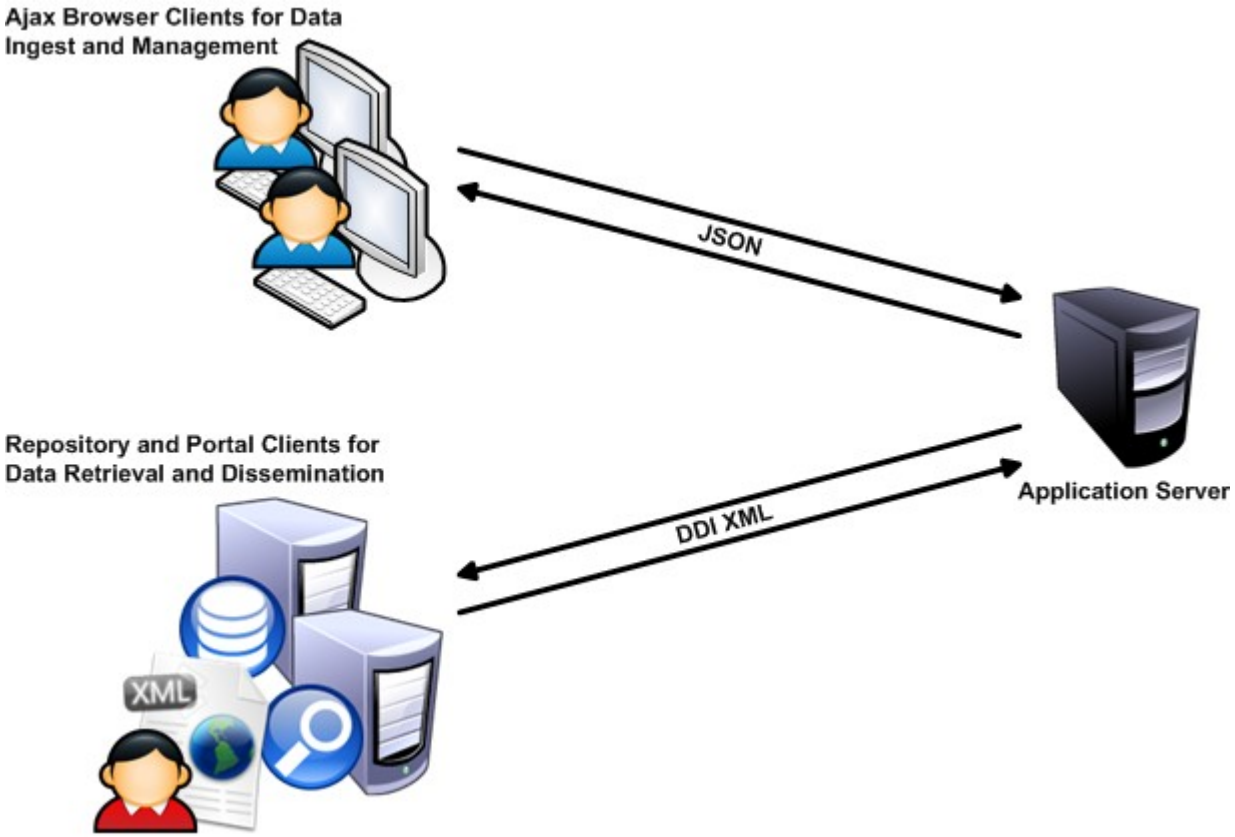
- Strategy
  - No ambition of finding a general solution
  - Approach „One class per complex type“
  - Approach „One class per element“

- General finding
  - Too many (join) tables without substantial content
  - Very few tables which hold all relevant information
  - Not intuitive types
  - Very time consuming and not promising

- Conclusion
  - Object-relational mapping close to DDI Schema creates a crude relational model
  - Early compromises abet early erosion of code
  - My paradigm now: Ensure that own classes lead to valid DDI

Basic Architectural Foundations from Prototyping

Communication between Clients and Server
Two Formats for Data Exchange



Basic Architectural Foundations from Prototyping

Communication between Clients and Server
Some Reasons for JSON

- Promise of relatively higher performance
- Easily and quickly changeable
- Presentation model differs from DDI data model
- Exchange between view-driven user interfaces does not be standardized
- Can or perhaps needs to contain user interface specific information
- Avoid binding of clients to a specific DDI version

Basic Architectural Foundations from Prototyping

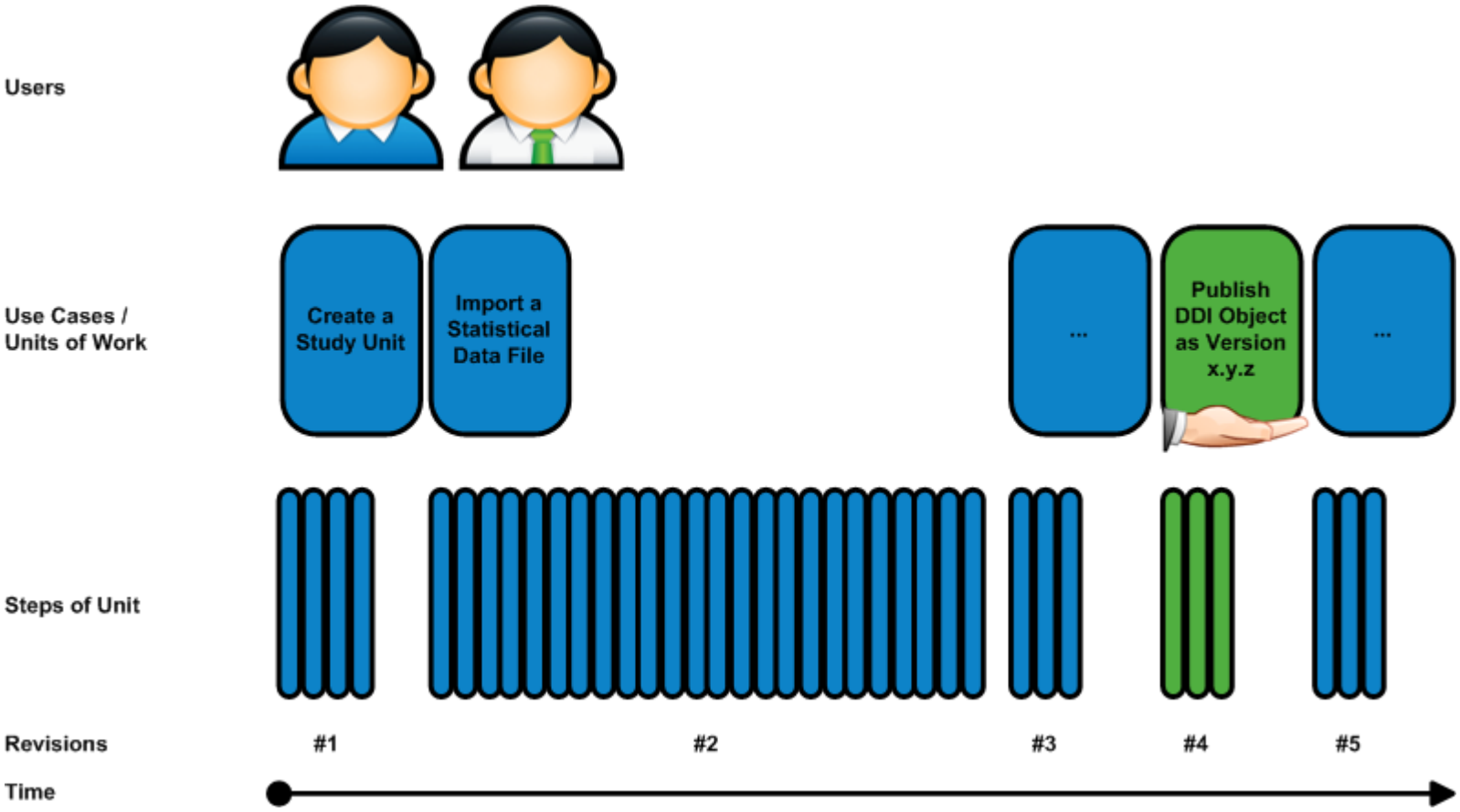
Concept of Historization and DDI Versioning
Defining the Difference

- Historization
  - Tracking of all changes of objects with their properties and assoziationss to other objects
  - Foundation for undo mechanism and object versioning
  - Characteristics: Lot of small data units, quick writing, many changes

- DDI Versioning (Publishing)
  - Ensure that a published maintainable can not be changed any more
  - Ensure fast access to a published maintainable
  - „Publish a mintainable DDI object“ means: Label it at a certain revision of the database with version number and as published
  - Not “labeled” revisions may be deleted for relief some time
  - Characteristics: Much less data as with historization, quick reading, no changes

Basic Architectural Foundations from Prototyping

Concept of Historization and DDI Versioning
Role-Based Transactional Revisions



Basic Architectural Foundations from Prototyping

Concept of Historization and DDI Versioning
Simple Example of Historization – Hibernate Envers API 1

```
@Before
public void setUp() throws Exception {

    Session session = Persistence.getCurrentSession();      // transaction 1
    Transaction transaction = session.beginTransaction();
    Study study = new Study();                              // class model is not ddi
    study.setTitle("Title of a stdy");
    session.save(study);
    transaction.commit();

    session = Persistence.getCurrentSession();              // transaction 2
    transaction = session.beginTransaction();
    Creator creator = new Creator();
    creator.setFirstName("Alexander");
    creator.setLastName("Mühlbauer");
    session.save(creator);
    study.setTitle("Title of a study");
    study.addCreator(creator);
    session.update(study);
    transaction.commit();

    session = Persistence.getCurrentSession();              // transaction 3
    transaction = session.beginTransaction();
    study.removeCreator(creator);
    session.update(study);
    transaction.commit();
}
```

Basic Architectural Foundations from Prototyping

Concept of Historization and DDI Versioning
Simple Example of Historization – Hibernate Envers API 2

```
@Test
public final void getRevisions() {

    Session session = Persistence.getCurrentSession();
    session.beginTransaction();
    AuditReader auditReader = AuditReaderFactory.get(session);

    long studyId = 1;
    List<Number> revisions = auditReader.getRevisions(Study.class, studyId);
    assertEquals(revisions.size(), 3);

    assertEquals(revisions.get(0), 1);
    Study study = (Study) auditReader.find(Study.class, studyId, revisions.get(0));
    assertEquals(study.getTitle(), "Title of a stdy");
    assertEquals(study.getCreators().size(), 0);

    assertEquals(revisions.get(1), 2);
    study = auditReader.find(Study.class, studyId, revisions.get(1));
    assertEquals(study.getTitle(), "Title of a study");
    assertEquals(study.getCreators().size(), 1);

    assertEquals(revisions.get(2), 3);
    study = auditReader.find(Study.class, studyId, revisions.get(2));
    assertEquals(study.getTitle(), "Title of a study");
    assertEquals(study.getCreators().size(), 0);
}
```

Basic Architectural Foundations from Prototyping

Concept of Historization and DDI Versioning
Simple Example of Historization – Hibernate Envers API 2

```
@Test
public final void forRevisionsOfEntity() {

    Session session = Persistence.getCurrentSession();
    session.beginTransaction();
    AuditReader auditReader = AuditReaderFactory.get(session);

    AuditQuery auditQuery = auditReader.createQuery().forRevisionsOfEntity(Study.class, false, true);
    List<Object[]> rersults = auditQuery.getResultList();

    Object[] result = rersults.get(0);
    Study study = (Study)result[0];
    DefaultRevisionEntity defaultRevisionEntity = (DefaultRevisionEntity)result[1];
    RevisionType revisionType = (RevisionType)result[2];
    assertEquals(study.getTitle(), "Title of a stdy");
    assertEquals(defaultRevisionEntity.getId(), 1);
    assertEquals(revisionType, RevisionType.ADD);

    result = rersults.get(1);
    study = (Study)result[0];
    defaultRevisionEntity = (DefaultRevisionEntity)result[1];
    revisionType = (RevisionType)result[2];
    assertEquals(study.getTitle(), "Title of a study");
    assertEquals(defaultRevisionEntity.getId(), 2);
    assertEquals(revisionType, RevisionType.MOD);
}
```

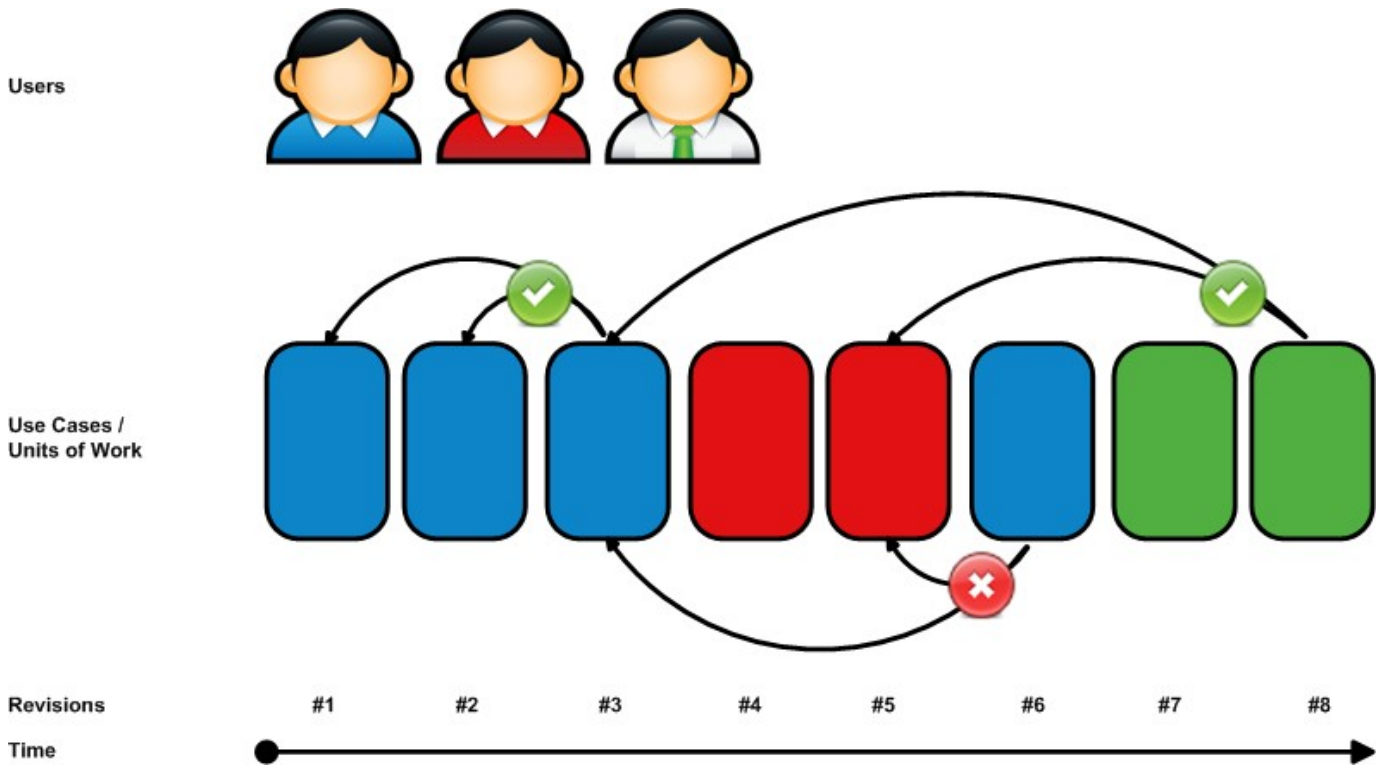
Basic Architectural Foundations from Prototyping

Undo Mechanism During Documentation Process

- Undo is a convenient feature supporting the documentation process
- Often demanded by users as self-evident feature
- Unfortunately, design and implementation is not as easy as one might think, the general requirement can quickly become very complex
- We defined an appropriate and well understandable undo scenario for our needs
  - User can undo own changes
  - Admin can undo own and other users´ changes
  - No redo (undo of undo)

Basic Architectural Foundations from Prototyping

Undo Mechanism During Documentation Process
Role-Based Linear Undo in a Multi-User Environment



Facing Various Challenges

- The plan was to have already finished a prototype.
- But there have been several challenges, the biggest are still:
  - To cope with appropriate technology stack
  - To neatly map and normalize existing data structures to DDI-L
  - To pore over DDI class modeling with suitable RDBMS persistence
- But we stay tuned! ☺

Thank you for your attention!

Any questions?

Alexander Mühlbauer
GESIS - Leibniz Institute for the Social Sciences
Data Archive for the Social Sciences

alexander.muehlbauer@gesis.org