BUILDING A MODULAR DDI 3 EDITOR

By Jannik Jensen and Dan Kristiansen with Alerk Amin, Arofan Gregory, Agostina Martinez, Martin Mechtel, Mary Vardigan, and Wolfgang Zenk-Möltgen

01/13/2010

DDI Working Paper Series -- Use Cases, No. 2

This paper is part of a series that focuses on DDI usage and how the metadata specification should be applied in a variety of settings by a variety of organizations and individuals. Support for this working paper series was provided by the authors' home institutions; by GESIS - Leibniz Institute for the Social Sciences; by Schloss Dagstuhl - Leibniz Center for Informatics; and by the DDI Alliance.

Building a Modular DDI 3 Editor

BY JANNIK JENSEN AND DAN KRISTIANSEN WITH ALERK AMIN, AROFAN GREGORY, AGOSTINA MARTINEZ, MARTIN MECHTEL, MARY VARDIGAN, AND WOLFGANG ZENK-MÖLTGEN

ABSTRACT

Developers at the Danish Data Archive (DDA) have built an open source DDI 3 editor using a layered architecture. The tool plays a critical role in the work flow at the DDA, providing needed integration of information. The design decisions made during development may be instructive for others building tools based on DDI 3.

BACKGROUND

For DDI 2, many people built tools and there was considerable duplication of effort. For DDI 3, it was decided that a more sensible approach was to pool resources and to share in the development of tools. The Foundation Tools Program was established in 2007 as a communal open source effort with stakeholders from several institutions contributing time and money to developing DDI 3-based tools. The program has thus far supported the creation of a set of DDI utilities and has contributed to the development of an editor, built primarily at the Danish Data Archive (DDA). The intention was to produce a workbench-style DDI editor architected in such a way that layers can be swapped out and replaced.

The Danish Data Archive (DDA), a national data bank for researchers and students in Denmark and abroad, is dedicated to the acquisition, preservation and dissemination of machine-readable data created by researchers from the social science and health science communities. The DDA has a need to convert existing OSIRIS and DDI 2 documentation to DDI 3 and to integrate information from various metadata sources across the life cycle. The editor is viewed as key to this integration.

The DDA first developed a common object model in Java for DDI 3 using the Apache XMLBeans technology. Around this object model the DDA has built tools for secondary validation and URN element generation and a mechanism to extract studies contained within a grouped structure. Work on the editor tool, which is being built using the Eclipse Rich Client Platform (RCP) as the front end, began in the fall of 2008.

USE CASE / REQUIREMENTS

The requirements for this project fall into three main categories.

DDI 3 specific needs

DDA is relying on other projects to capture metadata in DDI 3 format and thus is retrieving metadata from SPSS data files using tools like DExT and the statistical conversion tools built by GESIS. The DDI 3 requirements for the DDI editor project are to provide coverage from the variable level up to the study level and to

develop functionality to group time series and panel studies, drawing on the functionality of the tools developed in 2008.

Tools development

The tool developed must be open in order to permit further development. Thus it is a requirement to provide reusable tools/ components for the community under an open source license.

User perspective

Because the user perspective is critical, another important requirement is to hide the XML complexity of DDI 3 from the end user by providing a conceptual view of DDI3. This should be accessible through a GUI contained in a standalone application with the functionality for the community to hook into and add new features or specific functionality.

Information integration

Below is a diagram of the DDI system components that come into play during data acquisition, processing, and dissemination. This future state diagram highlights the role of the DDI editor in the data flow within DDA.

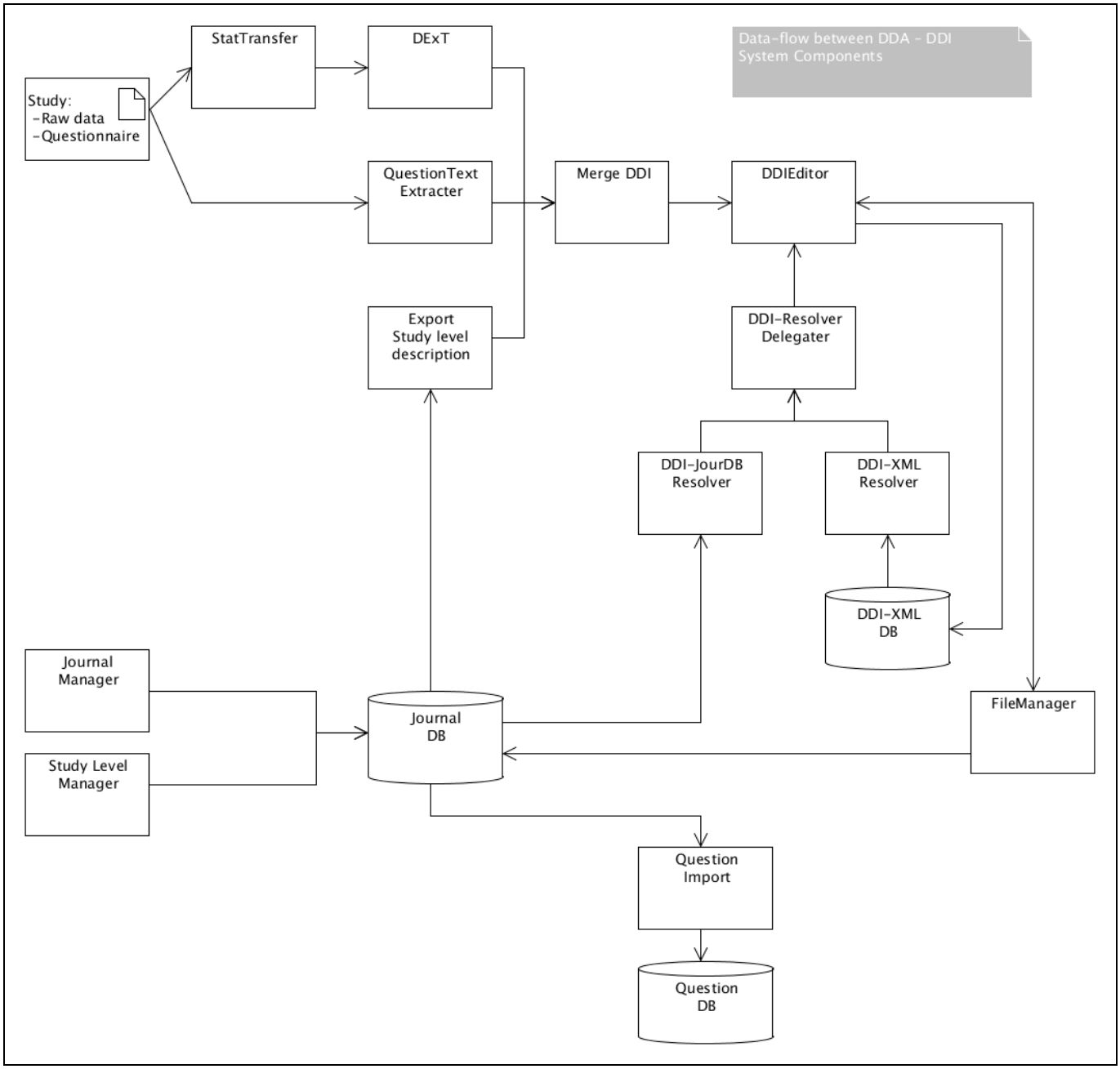

Figure 1: Future Data Flow at DDA

Inputs to the system include:

- Journal database containing study unit information
- Registry of shared resources, such as concepts, questions, and variables
- Outputs from conversion via DExT and other utilities
- XML generated from questionnaires

Outputs of the system include:

- DDI 3 XML
- Questions extracted and imported into a Question Database

The editor plays a central role in bringing information together. The system resolves all references before it stores them, but maintains a record of the original source. This is a requirement for long-term preservation.

POSSIBLE SOLUTIONS and DESIGN CHOICES

Several design options were considered along the way to building the first version of the editor.

Model and light model

The previously developed approach using only the DDI 3.0 XMLBeans object model raised issues with respect to the memory footprint because the XMLBeans read the full XML into memory and provide access to it via POJO, with getters and setters through DOM and parser functionality through SAX. Thus, rather than marshalling the whole DDI 3 XML into the object model, it was decided to marshal DDI 3 elements as needed. On a related point, it was decided to omit the complete DDI version number (e.g., DDI 3.1) from the naming structure embedded in the DDI model in order to minimize the need of code re-factoring in the event of new DDI releases.

Early in development it became clear that element selection for CRUD (create update delete) operations, needed for the Group inheritance structure, was not feasible without violating the minimization of the marshalling decision mentioned above. For example, to select a question for editing the whole question scheme had to be captured to list its available questions for selection.

Because of these issues, a generic light model was developed for more efficient processing. This focuses on the needed properties to create a selectable list of child elements contained by a parent element. The light model consists of a list of elements with zero to many labels. The element has attributes of type, ID, version, parent ID, and parent version. The label has a language attribute and string content. This means that it is possible to generate a list to select from without loading the full XML into memory.

Like the DDI 3 model the generic light model is implemented in Apache XMLBeans.

XML back end and unit of work design pattern

In order to retrieve and perform CRUD operations on DDI 3 XML without having to load XML into DOM or parse it via SAX, the developers tested several solutions:

- **Combining a DDI 3 object model via JAXB with ORM frameworks like Hibernate by Jboss, defined in the HibernateJAXB project.** As the complexity of DDI 3 is high and with its 612 elements in DDI 3.1 the outcome of applying HibernateJAXB on DDI 3 turned into too big a relational model to maintain performance.

- **A binary parsing tool for XML, VDT-XML by XimpleWare.** VDT-XML proved to be as fast as claimed but is lacking simple API CRUD operations. Thus API support for the project is available under the GNU GPL licence and a commercial license. VDT-XML was rejected due to licensing issues with other used projects licensed under the Eclipse EPL licence. As the commercial approach is not an option, VDT-XML is not feasible.

- **XML databases.** Several XML databases like eXist were tested based on their API functionality, usability, and performance. The Oracle Berkeley DB XML was selected as the default implementation. With the choice of a database as the backend came functionality such as indexes and transactions. The database had to fit within the licensing model of the project, which also pointed us toward the Berkeley DB. Benefits of the XML database approach include that such a database runs in a separate process, can handle large amounts of XML, and one can get chunks of XML back easily. Another benefit of native database XML is that when DDI is mixed with other standards, one only has to make queries to retrieve, e.g., Genericode, METS, etc.

Regarding transactions, a decision was made to implement the unit of work pattern as opposed to long-running transactions with the potential of locking off other CRUD operations.

DDI 3 indexing

The DDI standard is organised in modules – for example, Reusable elements, Data collection, Study Unit, etc. In XML terms these modules are called namespaces. XML as a standard does not define the application of namespaces to elements in one specific way. The namespace can either be declared as a prefix associated with a namespace at the top of the XML document or by defining a related namespace attribute on each element or as a mix of the two options. This complexity led to difficulties when querying and performing CRUD operations on various defined DDI XML because of the need to declare in which namespace any given element resides.

In order to embrace this complexity it was decided to inline all namespace declarations via the namespace attribute option in queries as well as in the XML output when DDI is exported out of the editor.

To implement this design decision, generic indexing of the DDI XML schema files was preferred over manual generation. This indexing scheme resulted in the generation of several indexes, listed below:

- A list of element-to-namespace relationships for all DDI elements
- A list of all abstract Identifiables containing element-to-Identifiable, -Versionable, or -Maintainable relationships
- A list of structural relationships among Identifiables, e.g., a question belongs to a specific question scheme and a question scheme belongs to the Data Collection module and the Data Collection module belongs to the Study Unit

When indexing the DDI XML schema files, another issue -- duplicate element naming – emerged, i.e., an element named Study Unit is specified in both the Study Unit module and in the Group module. This led to a revision of the indexing strategy to include duplicate element naming via a naming convention in the generated lists on the second occurrence of a previous defined element. The naming convention is defined as 'namespace__elementname' and is only used within the framework. The publicly available operations in the exposed API address element names as they are defined in the DDI XML schemas.

Indexing the DDI XML schemas also minimizes the burden of implementing new versions of the DDI specification by automatically adopting new or changed element definitions of future releases of the DDI-3.X standard.

Decoupling and façade controllers

The application design is based on an n-tier architecture consisting of presentation, logic, and model layers with a persistence layer at the bottom. The top presentation layer is decoupled from the lower layers. The rationale for this decision is to provide a DDI framework/application program interface (API) that other applications can tap into.

An additional design decision aiming towards reuse and further development is the endorsement of façade controllers among the components within the lower layers.

ISSUES / RESTRICTIONS

There are still some open issues to be resolved in the near future.

- **Retrieval of elements with same ID.** If and only if two parent elements have the same definition of ID and version plus the ID of the element itself holds the same value, the current design (retrieved by ID, version, parent ID, and parent version) cannot distinguish between the two, e.g.,

  us.icpsr:ConceptScheme.cs_1.1.0.0:Concept.c_1.1.0.0

  versus

  dk.dda:ConceptScheme.cs_1.1.0.0:Concept.c_1.1.0.0

  Additionally, agency has to be added to the implementation to distinguish between them.

- **DDI 3 Indexing.** Currently the relationships among Identifiables are indexed through a revised list (Appendix 1) of the Technical Specification-Part I of DDI documentation. This approach can lead to potential mistakes. A better way would be, as in the case of the other generated lists, a direct parsing of the DDI schema files themselves to retrieve structural information.
- **Support for XHTML**. The DDI Editor currently does not support XHTML in StructuredText DDI elements (currently no Open Source Eclipse RCP plug-in has been identified). It has yet to be decided whether a visual What You See Is What You Get (WYSIWYG) editor in RCP for the DDI-supported XHTML tags will be developed.
- **User access control.** User access control is not a requirement for DDA to build into the framework. However, the architectural design emphasis of using façade controllers is geared towards adding authentication and authorization features.
- **Referential constraints**. If, for example, the ID of a referenced Concept is changed, it is currently not possible to have the changed ID automatically reflected everywhere it is referenced. Checking referential constraints is an identified feature of secondary validation. This functionality can be used when implementing automatic updating of references upon a change of ID. It has currently not been decided to add this feature to the framework. The current design applies ID change constraints in the visual layer.

RESULTS

Below is a diagram of the layered architecture that resulted from the design decisions.

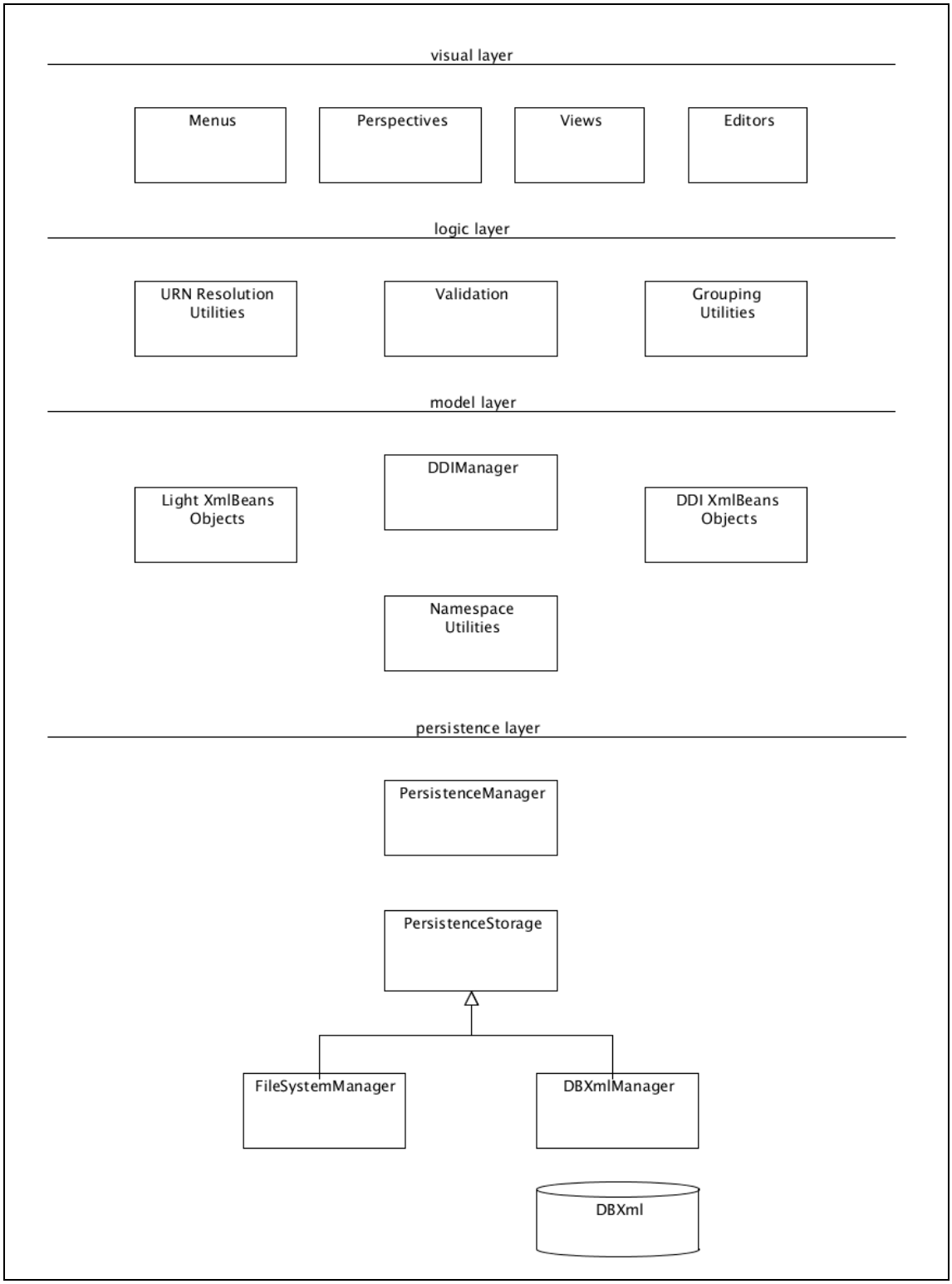

Figure 2: Architecture of Editor

The layered architecture consists of four layers. At the top is the user interface (UI), and the UI draws upon the lower layers. The lower layers are organized into three with a driving center layer being the model layer responsible for serving up objects to the other layers. The logic layer retrieves the DDI metadata from the model layer and performs various functions upon it. The model layer draws upon the functionality of the persistence layer and the persistence layer has the responsibility of handling the execution of the requests of the model layer. One can change the persistence layer. We describe the layers further below, from the bottom up.

Persistence layer

Within the persistence layer there are three main components with the DBXml database as the default persistent backend. The Persistence Manager is the façade controller of the layer and is responsible for delegating XQueries and their associated resources to its implementation manager defined by the interface Persistence Storage. When a resource is created or loaded into the framework, its persistence residence is recorded and it is preliminarily scanned for top-level DDI elements. Top-level DDI elements are Maintainable elements containing version or agency information.

The two default implementations of Persistence Storage are File System Manager and DBXml Manager. The File System Manager is responsible for interaction with the file system and the DBXml Manager is responsible for interaction with the DBXml database, executing XQueries and performing cursor scans, handling indexes on the DBXml database, and taking care of transactions.

The default persistence backend, the DBXml database, can be changed to, for example, the eXist XML database by development of an eXist Manager implementing the Persistence Storage interface. The same is possible for a relational database instead of an XML-based database, bearing in mind that the Persistence Manager parses XQueries on to the Persistence Storage implementation. In a relational database implementation the XQueries have to be parsed into SQL queries.

Model layer

The model layer has four components within it. It contains the model and the light model and it holds the DDI Manager, which is responsible for serving Java objects or XML through the generation of XQueries with the help of the namespace utility, as described in the section of this paper on DDI indexing design decisions. The model layer draws upon the service of the persistence layer to execute its generated XQueries.

Logic layer

The logic layer provides URN resolution and DDI validation as well as grouping functionality.

Visual layer

The visual layer is implemented in the Eclipse RCP. The visual layer provides UI elements for navigating the DDI structure as well as forms for viewing and editing specific elements. When working with Eclipse RCP, Perspectives can have one or more Views and/or Editors. The navigation panel at the left-hand side of the panel is a View and the right-hand panel is the Editor.

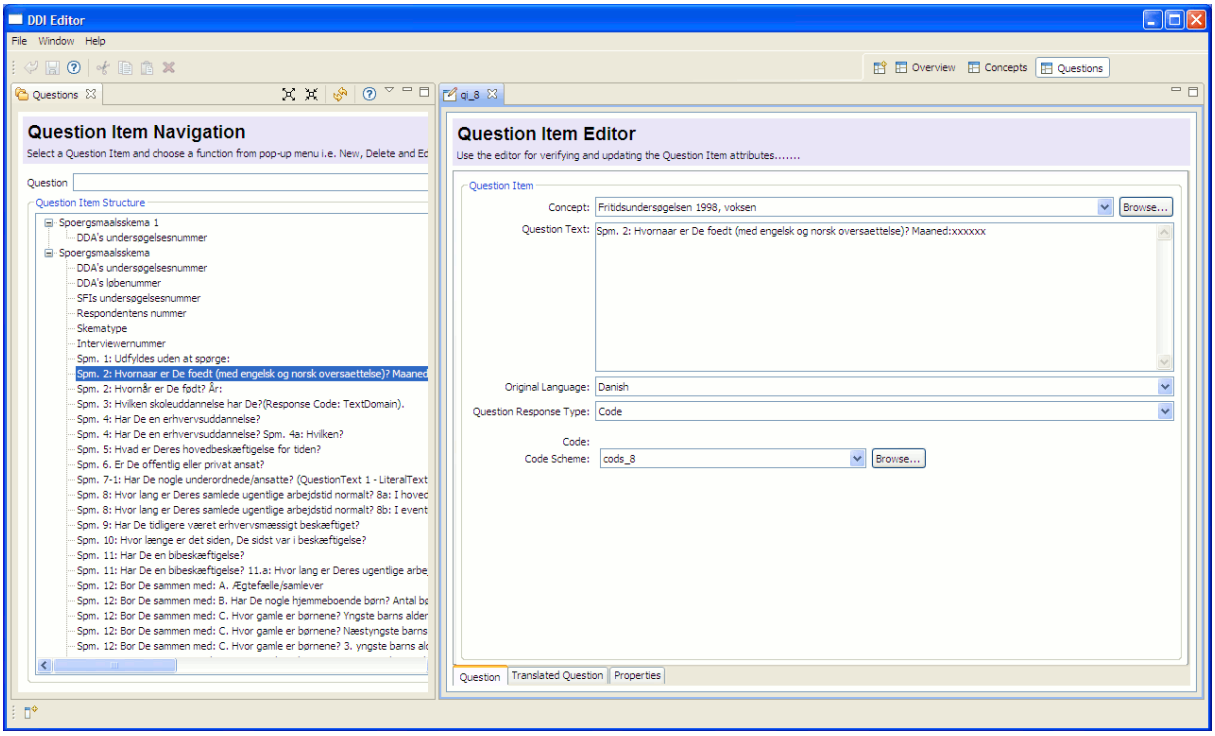

Figure 3: Visual Layout of the Editor (Perspective in RCP)

The API is implemented generically in the sense that adding additional methods for retrieving elements is easily done. If one had to write up specific methods for retrieving DDI elements each time, it wouldn't be a lot of work.

Our experience is that the XML is complex and that it takes some effort to query and update the XML files. The programming has taken 1.5 FTE for a year with contributions from the DDI Foundation Tools project to build core components and to review designs.

Deployment

Both the API and the UI are deployable through the Open Services Gateway initiative (OSGi) via the Eclipse Equinox implementation. The API can be deployed alone as standard JAVA packages or via OSGi. In both situations the deployment modules are the same and diagrammed below.

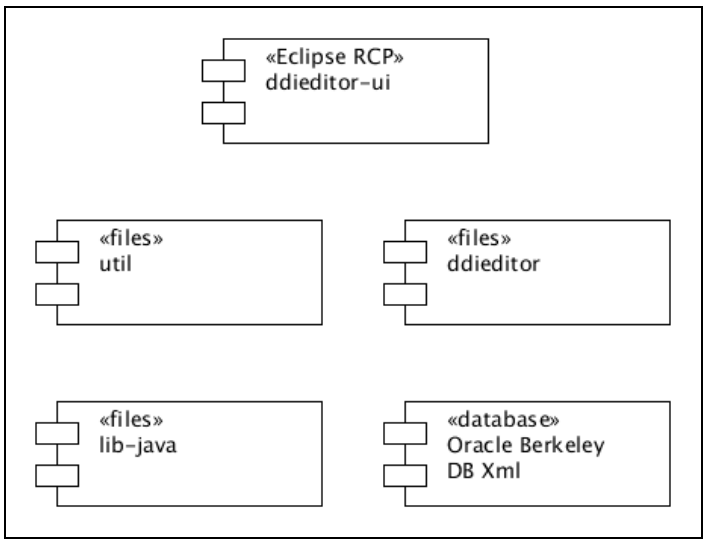

Figure 4: Deployment Diagram

The modules illustrated above constitute the Eclipse RCP application, the ddieditor-ui module. The util module is a further enhancement of the previously released util component of the DDI Foundation Tools project (DDI-FTP). The ddieditor module contains the API of the framework corresponding to the logic, model, and persistence layers outlined in the architecture section of the paper. The lib-java module is a resource module holding all resources needed by the other modules, including the Apache XMLBeans model of the DDI schemas and the light model described in the design choices section of the paper plus various needed library files of Open Source projects the framework depends upon.

XML Performance

The DDI editor as a whole is a very purely XML-centric application. The metadata are stored as XML, and queried and processed in XML. This has led to questions of performance with respect to the various processes of the application. The following is meant as a response to these questions and can be viewed as a preliminary practice for using the tools provided.

The processes of concern are the namespace injection to the queries, query resource delegation to implemented persistence storage, querying the DBXml database, and marshalling/unmarshalling from XML to the object model implemented via XmlBeans.

- Resolving namespaces and injecting them into the queries along with query resource delegation to the implemented persistence storage are both minimal as queries and resource relationships are cached.
- Querying the DBXml database is a more complex matter as the contents within the DBXml database are indexed on the attributes ID and version and this means a huge increase in query time as opposed to leaving, for example, the ID attribute out of the query. The result is a full table scan and a drastic increase in query time.
- The conversion of XML into the JAVA object model is dependent on the size of the XML on an exponential level. The rule of thumb is only to marshal the XML you need into XmlBeans, keeping your needs small. As the XmlBeans runs on the Java Virtual Machine (JVM), the XmlBeans are subject to the

overhead of being loaded into the JVM when used for the first time. Consecutive marshalling increases the time tenfold eventually until the loaded objects are garbage-collected by the JVM.

The design and implementation of the visual layer conforms as closely as possible to the above mentioned practices. Using the API for building other tools, the implementer has to take care since the API does not expose performance constraints on its use but focuses more on flexibility and openness.

OUTLOOK / CONCLUSION

Release date and status

Beta testing will be conducted in early 2010. An alpha version (of the API only) was made available at the end of 2009. See the project homepage for announcements and additional material: DDA DDI tools, http://samfund.dda.dk/dditools/default.htm. In 2011 DDA hopes to have implemented all the components in the revised data flow.

Licensing and maintenance

The tool will be licensed under LGPL, with full documentation of the tool to ensure that others can build on the tool effectively. Support from the DDI-FTP will be essential. There needs to be version control and a wiki in the ultimate location of the tool.

Future developments

Additional features may be added in the future. Comparison and on the fly group factoring are promising areas to develop. We would also like to integrate with registries to have resolution for referred elements. An enterprise edition of the framework would permit multiple users to work on the framework. The framework could sit at a central server.

Another important next step is to set the revised data flow into the context of the Open Archival Information System (OAIS).

APPENDIX A

The paper is one of several papers which are the outcome of a workshop held at Schloss Dagstuhl - Leibniz Center for Informatics in Wadern, Germany, November 2-6, 2009.

Workshop title:
Workshop on Implementation of DDI 3 - Advanced Topics

Organizers:
Arofan Gregory (Open Data Foundation, Tucson, Arizona, USA)
Wendy Thomas (Minnesota Population Center, University of Minnesota, USA)
Mary Vardigan (Inter-university Consortium for Political and Social Research [ICPSR], University of Michigan, USA)
Joachim Wackerow (GESIS, Leibniz Institute for the Social Sciences, Germany)
Link: http://www.dagstuhl.de/09452

This series was edited by Michelle Edwards, Larry Hoyle and Mary Vardigan.

The authors of the paper would like to acknowledge others who participated in this workshop.

Alerk Amin, CentERdata, Tilburg University, the Netherlands
Michelle Edwards, University of Guelph, Canada
Bryan Fitzpatrick, Rapanea Consulting, United Kingdom
Oliver Hopt, GESIS, Leibniz Institute for the Social Sciences, Bonn, Germany
Larry Hoyle, Institute for Policy and Social Research, University of Kansas, USA
Sanda Ionescu, Inter-university Consortium for Political and Social Research (ICPSR), University of Michigan, USA
Jannik Jensen, Dansk Data Archive (DDA), Denmark
Uwe Jensen, GESIS, Leibniz Institute for the Social Sciences, Köln, Germany
Mari Kleemola, Finnish Social Science Data Archive (FSD), University of Tampere, Finland
Dan Kristiansen, Dansk Data Archive (DDA), Denmark
Agostina Martinez, University of Cambridge, United Kingdom
Martin Mechtel, Institute for Educational Progress, Humboldt-Universität zu Berlin, Germany
Olof Olsson, Swedish National Data Service (SND), Sweden
Ørnulf Risnes, Norwegian Social Science Data Services (NSD), Norway
Wolfgang Zenk-Möltgen, GESIS, Leibniz Institute for the Social Sciences, Köln, Germany

APPENDIX B