

BEYOND FLAT FILES: DATA MODELLING, EDITING, ARCHIVAL AND INTERCHANGE

STEFFEN NEUMANN

Leibniz Institute of Plant Biochemistry, Department of Stress and Developmental
Biology, Weinberg 3, 06120 Halle, Germany

E-Mail: sneumann@IPB-Halle.de

Received: 12th June 2006 / Published: 31st August 2007

ABSTRACT

Software engineering today provides tools which minimize the need for manual coding of the typical components of an application, such as database, frontend and web application. Visual modelling brings together users and developers, and allows quick and direct communication about the topic. In the metabolomics community data models and XML formats for data interchange such as mzData are currently emerging. Using these standards as a show case, we present an infrastructure to support the use of these data standards and the process of getting there.

INTRODUCTION

Most communities in the Life Sciences are facing the problem of how to represent their data in a suitable way. The perfect data model should be flexible, to represent both standard and customized experimental set ups, stringent, to allow for validation and error-detection, machine readable, for storage and retrieval, open to ensure long-term archiving and accessibility, readable by the human eye, for debugging purposes – and of course easy to use. This contribution gives some experience of implementing software and the infrastructure for some emerging community data models.

In recent years metabolomics has become an important technology in solving functional genomics challenges [1] and mass spectrometry (both GC–MS and LC–MS methods) have been adapted to provide high throughput and broad coverage of metabolites [2, 3]. Large-

scale metabolomics experiments can produce huge amounts (up to 1 TB per machine per year) of raw data. Structured storage is the key to efficient access to the data for further processing and analysis. In addition to raw mass spectrometry data experimental meta-information is needed to match and compare results from different experiments. A standardized data exchange format allows community-wide collaboration and provides the basis for the large training sets needed in machine learning approaches.

Flat Files have been a commonly used storage model for biological data in the past years. For MS data exchange and as a vendor neutral format, both plain text peaklists or the (binary) netCDF format are being used. Both provide very little metadata – if at all – about the measurement set up, such as machine parameters, software used or by whom the experiment had been conducted. All of this information becomes important if the data is going to be archived for later (re-)processing. However, this requires parsers and converters for each client application processing the data.

Community-wide accepted data standards for interchange are currently emerging, such as mzXML[4] or mzData[5] in the context of the *Proteome Standards Initiative* (PSI). Converters from proprietary vendor file formats to mzData exist for e.g. Applied Biosystems, Bruker, Thermo Finnigan etc. For details see the web site of the Sashimi project¹ and the PSI². The *Architecture for Metabolomics* (ArMet) describes both metadata and results of metabolomics experiments [6], and is compliant with the recommended *Minimum Information about a Metabolomics experiment* (MIAMET)[7]. ArMet has been used in the Setup-X database [8]. All these emerging standards and data models can be used with current software engineering technologies.

The formalism of choice to describe these data models is a UML (class) diagram, which shows the “things” or more formally objects that are to be modelled. Examples in Fig. 3 are the User or a Peaklist. Each object has a set of named attributes of a given data type, such as Name of type String or Creation_Date of type Time Stamp in the example.

An *instance* of this data model consists of the set of objects with values assigned to the attributes. The purpose of a data exchange format is to allow transfer, without loss of information, to other pieces of software or even remote sites. The conversion process is also called serialization.

The *Extensible Markup Language* (XML) is a well-structured markup language. Content encoded in XML can easily be read by XML parsers, which exist for virtually any programming environment. An example of XML is shown in Fig. 1.

¹ <http://sashimi.sf.net/>

² <http://psidev.sf.net/ms/>

```

<admin>
  <sampleName>tt4_Batch1_1</sampleName>
  <sampleDescription comment="sampleNumber 1">
    <cvParam cvLabel="psi" accession="PSI:1000001"
      name="SampleNumber" value="Arabidopsis Seeds 1" />
    <cvParam cvLabel="psi" accession="PSI:1000006"
      name="SampleConcentration" value="2.57" />
    <cvParam cvLabel="psi" accession="PSI:1000002"
      name="SampleName" value="test sample" />
  </sampleDescription>
  <sourceFile>
    <nameOfFile>tt4_Batch1_1.wiff</nameOfFile>
    <pathToFile>file:/home/sneumann/data/</pathToFile>
    <fileType>Analyst QS 1.1</fileType>
  </sourceFile>
  <contact>
    <name>Christoph Boettcher</name>
    <institution>IPB Halle</institution>
    <contactInfo>+49 (0) 345 5582 0</contactInfo>
  </contact>
</admin>

```

Figure 1. XML excerpt of an mzData entry. Information is given either as an attribute (like cvParam) or in the body (like IPB Halle) of an attribute.

MODELLING

Regardless of which software development process (e.g. Waterfall or Extreme Programming) is adopted, during the early phase the purpose of the system needs to be defined. This can be done by describing typical *use cases* or requirements that the software has to fulfil. An example of such a use case for a repository system is given in Fig. 2.

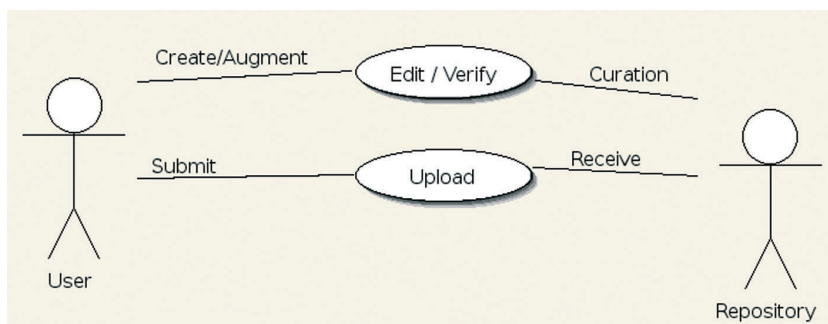


Figure 2. Use Cases for a repository system, with the user and the repository-institution as actors, and the two tasks edit/verify and upload as use cases. The connecting lines are annotated with the role an actor plays.

The actual data model should be created in close dialogue with the customers or users. A suitable model representation also for discussion are UML Class Diagrams. Initially, all “things” of interest should be collected, which will then usually end up as objects or entities in the final model. Next, their relationships have to be defined, such as “Experiment contains measurements” or “A Paper has one corresponding author”. The cardinality specifies that an author is *needed* for a paper, otherwise it will be rejected.

For data interchange, files need to be exported and imported on different current and future hardware (Intel, PowerPC) and operating systems (Windows, Unix and others) or sent over networks such as the internet, so the file representation has to ensure that any differences in encoding are either recorded for special treatment or that only the minimal consensus is used. XML is such a file format. The structure of the content can be described using either a *Document Type Definition* (DTD) or – more powerful in its expressiveness – an *XML Schema Definition* (XSD).

COLLABORATIVE DEVELOPMENT

For the success (or community-wide acceptance) of a data standard a large body of initial contributors and supporters is essential. The development should adopt the release-often-release-early approach also taken in many open source software projects, mentioned as one of the key points in Eric Raymonds essay “The cathedral and the bazaar” [9]. This will invite a broad range of comments and possibly fixes to the development version of a project.

This process of developing open standards and related open source software differs from commercial software development. Without the personal contact and meetings held in a company, there is a need for an efficient collaboration platform, which supports at least a code repository for sharing the current development, keeping the associated change-logs and allows release management. The other important task is to foster communication between the developers, hosting mailing lists (or equivalent functionality) with archives and search facilities.

The actual choice of platform depends on availability and personal opinion, and can vary between a general-purpose platform such as SourceForge³ or more targeted environments such as ProteomeCommons Project⁴

³ <http://www.sourceforge.net/>

⁴ <http://www.proteomecommons.org>

SAMPLE IMPLEMENTATION

The implementation is focused around the mzData model, since mzData has been created and described through a model in the *Unified Modeling Language* (UML) (see Fig. 3) and is available as XML schema. This description includes data types, classes, inheritance and constraints. First we describe the use cases for a simple mass spectrometry repository, then details on the third-party libraries and components are shown.

Use cases

The following use cases briefly define which actions should be supported by the infrastructure and applications for a MS repository. The six use cases underlying the implemented applications are:

Use Case1: Preparation for submission

A step which is necessary after an experiment has been performed, and the raw data has been converted to mzData. Depending on the converter, some fields might be filled with default or dummy values, such as `<institution\s) Not set </institution\s)` or `<cvParam cvLabel=\)psi\) accession=\)PSI:1000002\) name=\)-SampleName\) value=\)test sample\)/\s)` Such values need to be edited before uploading to a repository: the biologist loads the generated mzData file into an editor and checks the metadata. Once these have been corrected and fields added, the file needs to be verified against the schema and the defined constraints. If necessary, the file has to be edited until it passes validation.

Use Case 2.1: Submission of data

This involves both the biologist and the repository system. The biologist selects a file for upload to the submission form of the repository via a normal web browser. The repository validates the data against the schema and accepts or rejects the file. Finally, the data is persisted in the RDBMS.

Use Case 2.2: Batch import

Batch import is needed by the administrators if a large collection of data files need to be added to the database. A command line tool reads the files and persists them in the database.

Use Case 3: Curation of data

This is performed by the repository's curators, and is necessary if data needs to be changed after submission upon request, or to ensure the data quality. The curator connects to the database, selects an entry and reviews the corresponding values. Changes are persisted in the database, and a validation step guarantees consistency with the mzData schema.

Use Case 4: Browsing the repository

Allows members of the community to list and search the data in the public repository, and to download the corresponding XML file.

Use Case 5: Processing of stored MS data

A user can request this through the XCMS tool at the repository web site. After browsing the repository as described in Use Case 4, the (set of) mzData entries for processing is selected, and XCMS-specific parameters are adjusted. The raw MS signals are processed, retention times are aligned onto a common basis and the results are presented both in a tabular and graphical form.

SOFTWARE CHOICES

Creating such a large system would be impossible without (re-)using a range of third-party libraries and tools. In this section I describe those that we have chosen for our MetWare system.

The type of databases most commonly used today are *Relational Database Management Systems* (RDBMS). The data is stored in tables, where each row is an entry, having its attribute values in the columns. Data manipulation and queries are formulated in the *Structured Query Language* (SQL), which declares 1) which tables are used in a query, 2) how they are to be joined, and 3) which attributes are extracted.

The connection between the database and the clients is done though the *Java Database Connectivity* (JDBC) for Java based standalone clients or the Web frontend, or via *Open Database Connectivity* (ODBC) libraries for non-Java clients. These layers eliminate the need to use proprietary client APIs and wrap them if the database is exchanged for a different brand. Though the connection is vendor-independent, the SQL dialects are not, and common pitfalls exist when e.g. porting a MySQL query to Oracle. Another layer of indirection introducing database adapters can convert generic query statements into the vendor-specific dialect.

MODEL DRIVEN ARCHITECTURE

In a Model Driven Architecture the data model is defined as a *Platform Independent Model* (PIM) and afterwards transformed into a *Platform Specific Model* (PSM) for a specific architecture and language.

The Eclipse Modelling Framework⁶ provides code generation facilities for Java classes implementing the model, adapters for viewing, change notification and undo capabilities and a basic editor with validation against the model schema. EMF has been used to import the mzData model and create the model implementation and editor. However, the EMF itself does not provide database persistence.

⁶ <http://www.eclipse.org/emf/>

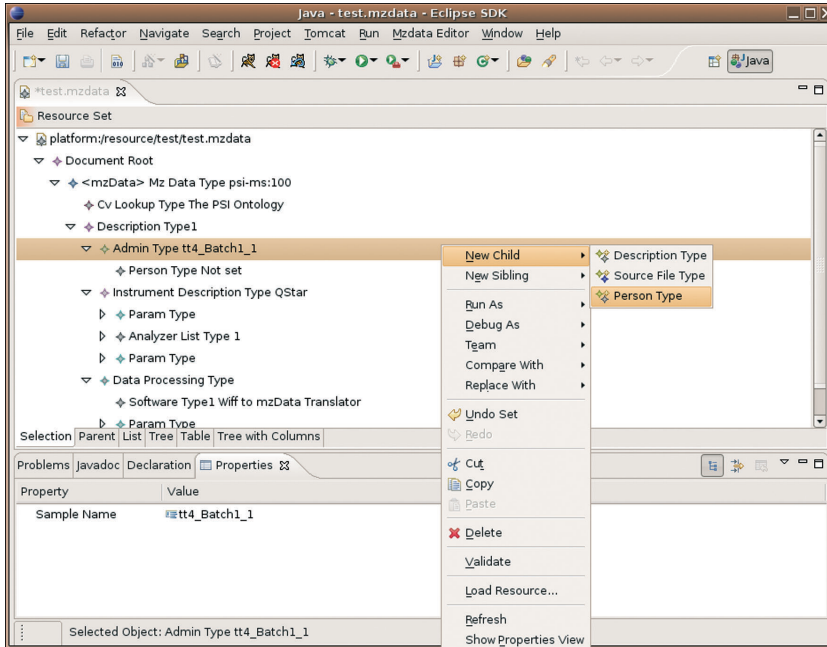


Figure 4. Generated Editor for mzData. The entry is shown as a tree-view, with properties (values) of the tags shown at the bottom of the window. Context sensitive menus provide schema-compatible insertion of children or siblings and the verification of a (sub-)tree.

The persistence of the EMF objects is handled through an object relational mapping. *Java Data Objects* (JDO) from Sun⁷ offer access to different data stores and manage transactions. Persisted data can be queried and transformed into native Java programming language objects. JPOX⁸ is the reference implementation of the JDO2.0 specification and can attach to most available relational databases. The eclipse plugin from Springsite⁹ generates the metadata for JDO and integrates code that readily allows the editor frontend to be used on data stored in the database.

The presentation layer of the web application is implemented using *Java Server Faces* (JSF) from Sun¹⁰, which provide the framework for handling user sessions, lifecycle of backing objects and navigation between the pages. JSF Tag libraries provide additional widgets which can be used to present tree views, show popup help or integrate a layout templating engine.

⁷ <http://java.sun.com/products/jdo/>

⁸ <http://www.jpox.org/>

⁹ <http://elver.org/>

¹⁰ <http://java.sun.com/javaee/javaserverfaces/>

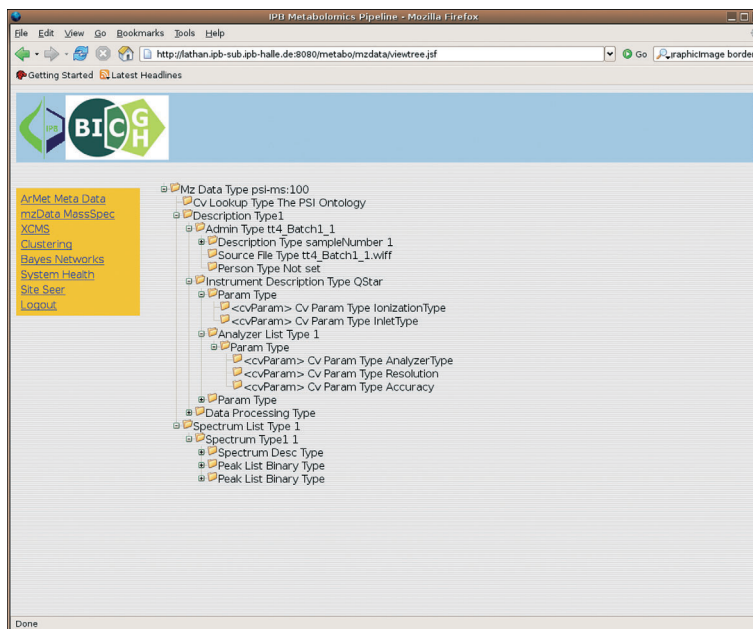
DATA PROCESSING

For analysis software written in Java that can readily incorporate and use the JDO libraries, data access can be done in the *JDO Query Language* (JDOQL), similar to the SQL query language.

For signal processing tasks mentioned in Use Case 5 (alignment of retention time shifts and higher level analysis) we integrate a backend service using the XCMS package from the statistics software R and Bioconductor [11] project. XCMS performs peak picking, retention time alignment of multiple LC-MS or GC-MS runs and generates a list of differential mass signals. Communication between R and the application server is done via the Rserve protocol¹¹. To connect XCMS to the database backend, we created SQL queries which retrieve the binary data from the RDBMS and feed it into the modified mzData parser. For a detailed description of XCMS see [12].

RESULTS

We have focused on the creation of the backend storage and applications for the use cases Use Case 1 to Use Case 3. In the following paragraphs we describe first experience with the implementation.



¹¹ <http://stats.math.uni-augsburg.de/Rserve/>

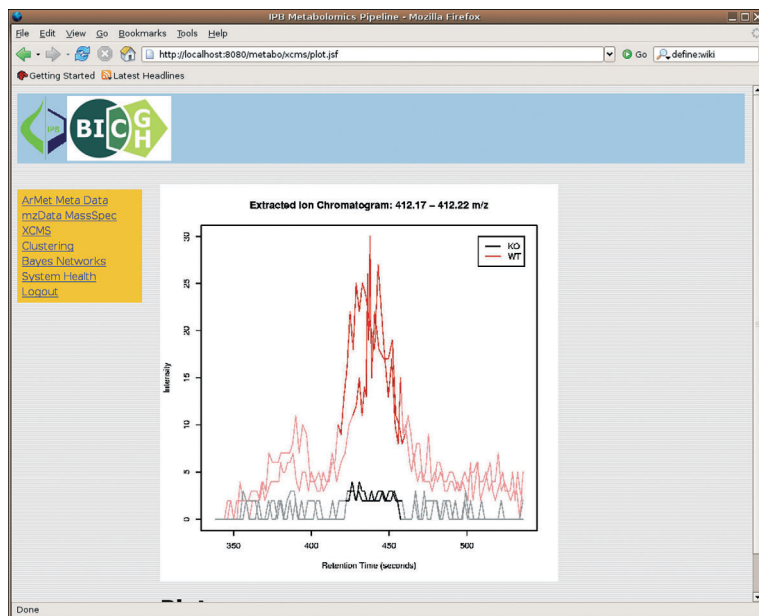


Figure 5. The web interface for the mzData model, showing a part of the tree view. Example of XCMS output: aligned raw data for a differential mass signal.

The editor for Use Case 1 (preparation of mzData XML files for submission) was the first to be finished using EMF. A screenshot is shown in Fig. 4. It can easily handle data files of around 100 MB, which had been acquired on our LC-MS set up using an Applied Biosystems QStar mass spectrometer and were transformed from the instrument-specific wiff format to mzData with a vendor supplied converter. The validation of said 100 MB file is completed in less than a second and is no additional burden to the biologist. The editor can be downloaded at <http://msbi.ipb-halle.de/>.

The persistence enabled Editor used for Use Case 3 (Curation) that connects to the RDBMS via JDO offers the same functionality as the standalone version. Since lazy loading is implemented, only the relevant parts of the data are requested from the database. Even large collections can be accessed this way.

The web-system is currently being evaluated and improved to provide a biologist-friendly user interface design for the outlined use cases Use Case 4 and Use Case 5, with modules (see Fig. 5) existing for both of them. The architecture of the system (application-, R-statistics- and database server) allows for an easy integration of high-level analyses. Prototypes for these modules are included in the web application.

CONCLUSION

The chosen data standards are currently gaining a wider acceptance in the metabolomics community. A flexible software development process is necessary to accommodate frequent changes without the need for manual adaption of the resulting software. The overall system consists of the database, R server and web application server, all of which can run on different machines. To scale to a large number of concurrent users, all three services can be run on a cluster of machines, sharing the load. A common filesystem layout is not needed.

We provide the service to biologists working in our institute and close collaborators. A demo database is available at <http://msbi.ipb-halle.de/>. In the future we plan to implement a similar system for the ArMet metadata, and tight integration of externally controlled vocabulary and ontologies.

Projects starting a standardization effort should consider modelling their data on a public platform and invite other parties to comment or even participate. Getting the actual model “right” (flexible, stringent, machine-/human-readable and easy to use) can be expected to be the hardest task. The standard should be closely followed by software implementing data capture and handling, with the database access coming last. The MDA approach makes it possible to recreate the necessary code basis and backend database with minimal manual coding, since the data standard is hopefully going to evolve.

ACKNOWLEDGEMENTS

Nigel Hardy, Helen Jenkins, Chris Taylor, Kai Runte and many others for the ArMet and MzData models. Dierk Scheel, Jürgen Schmidt for their valuable discussions on mass spectrometry and biology.

The work is supported under BMBF grant 0312706G.

REFERENCES

- [1] Goodacre, R., Vaidyanathan, S., Dunn, W.B., Harrigan, G.G., Kell, D.B.(2004) Metabolomics by numbers: acquiring and understanding global metabolite data. *Trends Biotechnol.* **22**(5): 246–252..
 - [2] Roepenack-Lahaye, E.v., Degenkolb, T., Zerjeski, M., Franz, M., Roth, U., Wessjohann, L., Schmidt, J., Scheel, D., Clemens, S. (2004) Profiling of Arabidopsis secondary metabolites by capillary liquid chromatography coupled to electrospray ionization quadrupole Time-of-Flight mass spectrometry. *Plant Physiol.* **134**:548–559.
-

- [3] Roessner, U., Wagner, C., Kopka, J., Trethewey, R., Willmitzer, L. (2000) Technical advance: simultaneous analysis of metabolites in potato tuber by gas chromatography-mass spectrometry. *Plant J.* **23**:131–142.
- [4] Pedrioli, P.G.A., Eng, J.K., Hubley, R., Vogelzang, M., Deutsch, E.W., Raught, B., Pratt, B., Nilsson, E., Angeletti, R.H., Apweiler, R., Cheung, K., Costello, C.E., Hermjakob, H., Huang, S., Julian, R.K., Kapp, E., McComb, M.E., Oliver, S.G., Omenn, G., Paton, N.W., Simpson, R., Smith, R., Taylor, C.F., Zhu, W., Aebersold, R. (2004) A common open representation of mass spectrometry data and its application to proteomics research. *Nature Biotechnol.* **22**(11):1459–1466.
- [5] Orchard, S., Hermjakob, H., Binz, P., Hoogland, C., Taylor, C., Zhu, W., Julian, R.J., Apweiler, R. (2005) Further steps towards data standardisation. *Proteomics* **5**(2):337–339.
- [6] Jenkins, H., Hardy, N., Beckmann, M., Draper, J., Smith, A.R., Taylor, J., Fiehn, O., Goodacre, R., Bino, R.J., Hall, R., Kopka, J., Lane, G.A., Lange, B.M., Liu, J.R., Mendes, P., Nikolau, B.J., Oliver, S.G., Paton, N.W., Rhee, S., Roessner-Tunali, U., Saito, K., Smedsgaard, J., Sumner, L.W., Wang, T., Walsh, S., Wurtele, E.S., Kell, D.B. (2004) A proposed framework for the description of plant metabolomics experiments and their results. *Nature Biotechnol.* **22**(12):1601–1606.
- [7] Bino, R., Hall, R., Fiehn, O., Kopka, J., Saito, K., Draper, J., Nikolau, B., Mendes, P., Roessner-Tunali, U., Beale, M., Trethewey, R., Lange, B., Wurtele, E., Sumner, L. (2004) Potential of metabolomics as a functional genomics tool. *Trends Plant Sci.* **9**(9):418–425.
- [8] Fiehn, O.S.M., Wohlgemuth, G. (2005) Automatic annotation of metabolomic mass spectra by integrating experimental metadata. In: *Proceedings of DILS 2005*, no. 3615 in Proc. Lect. Notes Bioinformatics, pp.224–239. Springer.
- [9] Raymond, E.S. (1999) *The Cathedral and the Bazaar*. O'Reilly & Associates, Inc., Sebastapol, CA, USA.
- [10] Orchard, S., Taylor, C., Hermjakob, H., Zhu, W., Julian, R., Apweiler, R. (2004) Current status of proteomic standards development. *Expert Rev. Proteomics* **1**(2):179–183.
- [11] Gentleman, R.C., Carey, V.J., B DM, Bolstad, B., Dettling, M., Dudoit, S., Ellis, B., Gautier, L., Ge, Y., Gentry, J., Hornik, K., Hothorn, T., Huber, W., Iacus, S., Irizarry, R., Leisch, F., Li, C., Maechler, M., Rossini, A.J., Sawitzki, G., Smith, C., Smyth, G., Tierney, L., Yang, J.Y. H., Zhang, J. (2004) Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biol.* **5**:R80, [<http://genomebiology.com/2004/5/10/R80>].
- [12] Smith, C., Want, E., O'Maille, G., Abagyan, R., Siuzdak, G. (2006) XCMS: Processing mass spectrometry data for metabolite profiling using nonlinear peak alignment, matching and identification. *Analyt. Chem.* **78**(3):779–787.
-