DE NOVO MARKUP LANGUAGE, A STANDARD TO REPRESENT *DE NOVO* SEQUENCING RESULTS FROM MS/MS DATA

Savas Takan[1] and Jens Allmer[2]

[1]Computer Engineering, [2]Molecular Biology and Genetics
Izmir Institute of Technology, Gulbahce Campus, Urla, Izmir, Turkey
phone: + (90) 232 750 7517, email: jens@allmer.de
web: http://bioinformatics.iyte.edu.tr

ABSTRACT

Proteomics is the study of the proteins that can be derived from a genome. For the identification and sequencing of proteins, mass spectrometry has become the tool of choice. Within mass spectrometry-based proteomics, proteins can be identified or sequenced by either database search or *de novo* sequencing. Both methods have certain advantages and drawbacks but in the long run we envision *de novo* sequencing to become the predominant tool. Currently, *de novo* sequencing results are stored in arbitrary file formats, depending on the developers of the algorithms. We identified this as a large and unnecessary obstacle while integrating results from multiple *de novo* sequencing algorithms. Therefore, we designed a standard file format for the representation of *de novo* sequencing results. We further developed an application programming interface since we identified the lack of proper APIs as another obstacle, introducing a needlessly high learning curve for developers.

1. INTRODUCTION

Mass spectrometry (MS) has been widely used to investigate proteins and it is defining the field of proteomics today . Mass spectrometers can produce large amounts of measurements which need to be analysed computationally to infer meaning. Typically database search methods are employed to identify proteins from complex mixtures . However, often databases are not available or despite their availability some sequences are not readily found therein . To overcome this problem *de novo* sequencing can be used to directly produce a peptide sequence from a MS/MS spectrum for which many algorithms have been developed .

Currently, *de novo* prediction algorithms store all the information in their particular, if not peculiar, formats. Algorithm-specific solutions are abundantly available, but there is no a general solution to store and share *de novo* sequencing results which decreases their usefulness. Since they need to be integrated and further analysed and cannot be used directly as evidence for protein identification or sequencing at present standardization of *de novo* sequencing outputs is vitally important.

Several areas of mass spectrometry-based proteomics have seen standardization such as for collections of mass spectra and MS/MS spectra in files like mzXML , mzData and most recently in mzML . The Human Proteome Organization (HUPO) oversees many standard initiatives and is adopting community standards or actively develops new ones. An example for community standards are protXML and pepXml which store database search algorithms' results. HUPO is currently working on the mzIdentML standard to supersede pepXML and protXML but has not published it yet.

As there is currently no standard to represent *de novo* sequencing results, different, and in our opinion, unsuitable formats are used to store *de novo* predictions. PEAKS, a commercial *de novo* sequencing tool , for instance, stores results in pepXML format. Other *de novo* sequencing algorithms, for instance the two most popular freeware solutions, PepNovo and Lutefisk , use custom formats which unfortunately do not provide adequate information for further downstream analyses and knowledge mining. We believe it is necessary to develop a standard representation for *de novo* sequencing results since none of the current programs represent the minimum information needed and standards from related fields are either flawed or not fully applicable.

In *de novo* sequencing making a prediction based on multiple spectra is becoming more important . In the light of this knowledge, a serious design limitation of the proposed pepXML format is that each result can only be linked to one spectrum source. PepXML and mzXML files can become enormous (sometimes several gigabytes). This does ease the exchange of data but moves the problem of standardization to a technical difficulty. Since as many as 90% of spectra may remain unassigned, two options are viable to overcome the problem; one to link to specific spectra in an mzML or mzXML file and the other to include the spectrum in the exchange format. This leads to the possibility of exchanging only one file containing only relevant information. Automatic merging of documents, although not a vital feature, is currently not supported for any of the standards. PepXML was designed with the representation of the best result in mind. For *de novo* prediction this is a limitation since the search space is much larger than for database search algorithms so that multiple, equally probable results are frequent and need to be represented alongside each other to enable further analyses. Although around 100 post translational modifications are represented in current standards the user is expected to provide possible modifications. We believe that a format to represent search results in a database like pepXML should model only those results but instead information is inflated by many results of in-house tools provided by the developers of the standard like XPRess , ASAPRatio and PeptideProphet . Interestingly, it would be possible to store the results of PeptideProphet, for example, in pepXML, without it

978-1-4673-0878-6/12/$31.00 ©2012 IEEE

HIBIT April 19-22, 2012 Cappadocia, Turkey

being explicitly modelled, as it is now. Another issue along these lines is that a data model may never include visualization instructions, but the pepXML format includes such information as well as information which could easily be calculated from data modelled in the format and hence need not be stored.
In an attempt to overcome all these limitations, we set forth to develop the *de novo* markup language (DNML). The format that we propose here is meant as a standard for the representation of *de novo* sequencing results. First and foremost, we ensured that all necessary and as little unnecessary data as possible are represented in the format. Another issue that we identified which is crucial to the acceptance of a standard initiative is the availability of application programming interfaces (API) to read and write the standard. Many standards fail to offer an API and if they do only for reading of the standard. To increase the acceptance of DNML we went well beyond this and offer reading, writing, and conversion facilities. But most importantly, our API allows for the creation of files and can thus be used directly in any software that performs *de novo* sequencing. This removes the need to develop reading and writing functionality for all developers of *de novo* sequencing algorithms, a mundane task which is often neglected and therefore often produces peculiar outcomes. Finally, we also include reading of mzXML and mzML information in the API so that any developer of *de novo* sequencing software can completely focus on their algorithm. In the following, we will first introduce the schema and then the API. Finally, we represent our future direction.

2. METHODS

For programming, the Java™ language was selected because it seamlessly allows interoperability, has many libraries and has a huge support community. The Eclipse IDE was selected on the basis of available plug-ins and extensive user support. SVN, as provided on Origo , was used during development as a version control system. Maven in order to facilitate project integration, eUML was utilized to support UML visualisation during the developing process. Visual Paradigm was used to create some UML diagrams and to enable round-trip engineering.
Oxygen was used for extensible markup language (XML) editing because of the availability of an academic version. SAX and JAXB libraries for JAVA were used for parsing XML data. JAXB is a fast XML parser and since it provides SAX parsing also large files could be parsed. The XJB tool was used to provide class transformation from XML. Regular expressions were used in order to parse some textual information.

3. XML SCHEMA

The W3C developed document type definitions (DTDs) and XML schema languages to allow the validation of XML formatted documents. Since today DTDs seem to be deprecated, we developed a pure XML schema without any reference to DTDs. During the creation of our XML schema, object orient programming techniques were used for solving repeated, non-extendable data problems.

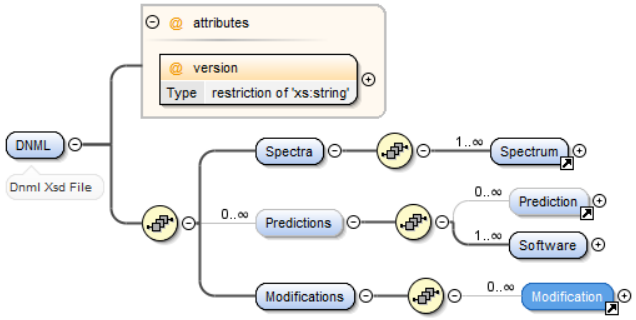

Figure 1: Depicts the DNML root element and the contained Spectra, Predictions, and Modifications elements. The XML tree can be further expanded to reveal that Predictions, for example, contain Prediction elements and a Software element.

The root of the DNML schema and its directly contained elements are shown in Figure 1. Spectra represent the underlying data source for *de novo* predictions and are modelled in two different ways, one to minimalize file size, and one directly linking to spectra contained in files of the appropriate standard. Because files in mzXML and mzML format quickly become huge, it is difficult to share them and therefore spectra can be contained within the DNML standard optionally. Different data encodings are provided to additionally reduce the size of the DNML file.

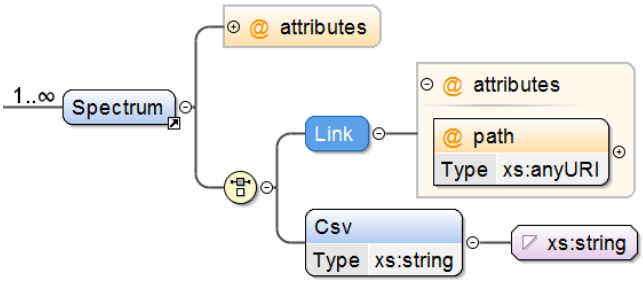

Figure 2: Shows how the raw data source, the spectrum, is modelled. Each spectrum has attributes such as id, precursor m/z, precursor intensity (not shown). The peaks in the MS/MS spectrum can either be provided as a link to a file containing spectra in mzML or mzXML standard, or as contained data.

DNML offers the opportunity to store spectrum data in several formats, such as Base64, comma separated values (CSV) and as links to already existing file formats. Our API further allows the future incorporation of additional data formats such as compressed formats due to extensible design (Figure 2).
When a spectrum is provided as a link to mzXML or mzML formatted files, the DNML file remains lean and the API allows retrieving the linked data.
While developing DNML the problem of merging multiple files has been taken into consideration. In the library, com-

mands are available that enable merging and splitting of DNML formatted files.

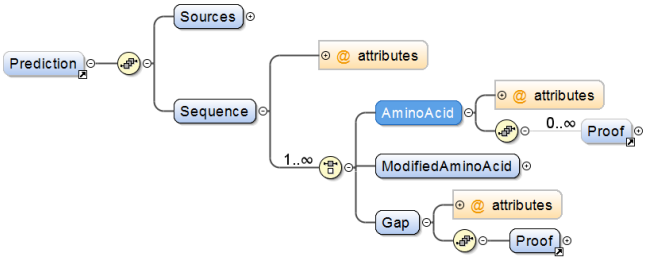

Figure 3: The Prediction element contains two elements, Sources and Sequence. Sources element defines which spectra led to the prediction. Sequence element has sequence and confidence attributes. A sequence can consist of amino acids, modified amino acids and gaps which allows object oriented modelling of a sequence. It also allows positional confidence and proofs for the sequence.

Our DNML format is the first one which allows a many-to-many mapping between predictions and spectra. This has become important since many *de novo* sequencing algorithms now use multiple spectra from different sources to produce a better prediction (Figure 3). Since a prediction can contain stretches where no amino acids can be assigned, it is necessary to model gaps. Our DNML standard is the first which enables this, but we take this a step further by enabling object oriented creation of sequences which are composed of three different types of elements, amino acids, modified amino acids and gaps. An additional novelty is that each sequence element can be amended with peaks proving the existence and with a positional confidence.

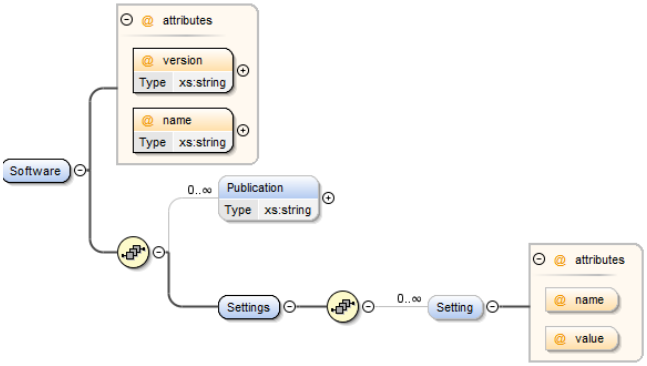

Figure 4: Software represents the tool that made one or multiple *de novo* predictions. Publication element is used to add publications of that software since we sometimes experienced trouble when searching for the first publication of an algorithm.

The user's software settings need to be provided since different settings can lead to significantly different predictions. Our model allows different Predicitions to have different settings so that the information is not repeated for each Prediction (Figure 4).

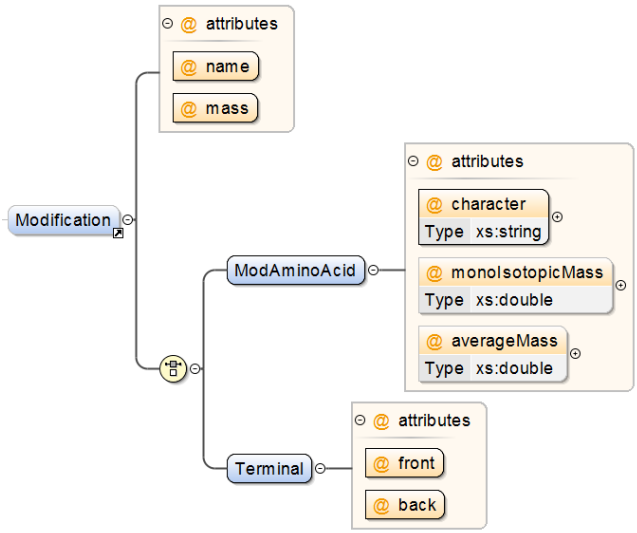

Figure 5: A post translational modification (PTM) is modelled as a change to an amino acid and a file with all currently known PTMs is provided to simplify handling of PTMs. Terminal modifications for peptides are also possible.

The proper modelling of post translational modifications (PTMs) is important today but will increase in importance in the near future when more algorithms will become able to predict unexpected PTMs. We allow user specified PTMs but also provide a list of about 100 known modifications. Another important innovation of DNML is that predictions can be complemented with the underlying proof for the sequence, something that is usually modelled as a simple score, is much better reflected in our format (Figure 3). The proof which is not completely expanded in Figure 3 links to the peaks in the MS/MS spectrum which define a particular sequence element. A sequence element can be an amino acid (normal), a modified amino acid, or a gap which means an unknown range in the MS/MS spectrum which was not assigned an amino acid (Figure 5).
The DNML schema is located at (https://svn.origo.ethz.ch/dnml/data/) and can be used for validating files in the format. The API we provide also comes packaged with the DNML schema for faster validation. In the following we will briefly introduce the API and our design choices for the underlying code.

4. PARSER SELECTION

We paid particular attention to using standard JAVA™ libraries for XML parsing for compatibility reasons.
Firstly, existing parsers were examined and two parsing method emerged as possibilities. One of these, uses the document object model (DOM) was found unsuitable since many result files are large and since DOM requires the entire file to be in memory after parsing. The simple API for XML (SAX) allows sequential access to XML elements and has a lower memory footprint and was therefore selected for this project. Specifically, the JAXB technology has been selected for XML parsing in our DNML API. XML mapping with the

XJB tool allowed for easy conversions between XML and Java classes and sped up the design process by reducing the duplication of code.
Two different libraries are used for parsing mzXML files. The first is the Jrap library which caused many problems for us but which enables us to retrieve spectra modelled as links in DNML format from the mzXML format. The other is the Jmzreader library which uses the JAXB JAVA™ technology which integrated seamlessly and provides the link between DNML and mzML.
Regular expression, part of the standard JAVA™ library, were mostly needed for conversion facilities for example to parse LutefiskXP and PepNovo *de novo* predictions. All parsers, for DNML, mzXML, mzML, existing files are designed such that the API can easily be extended with further parsers, not even requiring re-building of the code but allowing run-time binding of functional additions. The overall API design is completely built with this type of modularity in mind and will be briefly introduced in the following.

5. SOFTWARE MODEL

The DNML library provides an application programming interface to files written in the standard. Adding, deleting, merging, transferring data, and converting between selected standards are supported by the API. In addition to this, service, command classes can be added at runtime by using the Java resource bundle which creates great flexibility and makes the library easily extendible and modular (Figure 6).

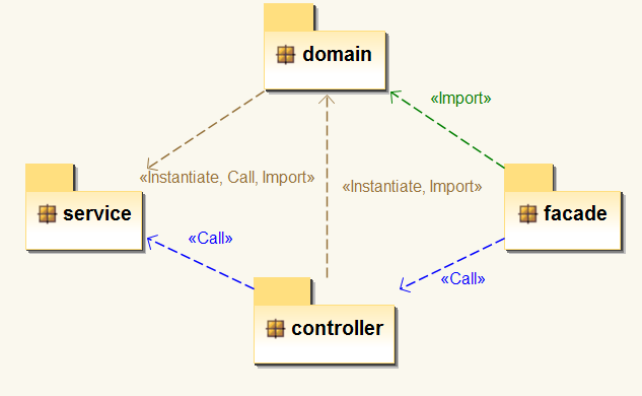

Figure 6: The DNML library has four packages. The façade package handles input and sends it to the controller. The controller finds, creates, and executes commands. The service package handles domain tasks.

The model view controller (MVC) approach was used to develop the DNML library because it helped us to achieve a modular and extensible architecture. The structure of the API has several layers, Façade, Controller, Services, and Model (Figure 6). Some of the topics that we deem important for modularity are explained briefly in the following, but additional information can be found in our online documentation at http://bioinformatics.iyte.edu.tr/dnml.

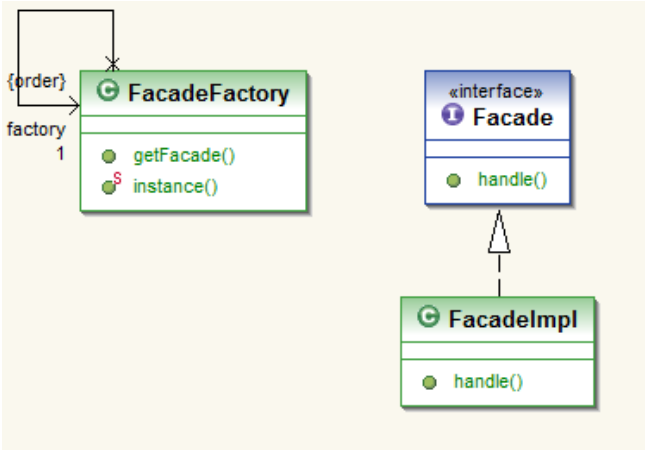

Figure 7: The façade package is the main interface. The factory singleton can only be accessed by the instance function. The façade factory uses getFacade to create a new façade which is implemented by FacadeImpl class. The handle function sends input to the control package which returns the library results.

The façade factory, a static singleton, is the main interface to the façade layer and provides the only means to communicate with the layer. Inputs are taken as a string array. In this way, the program can run from the console. When data is entered, the façade layer forwards inputs to the control factory which creates a command. The façade's output is a DNML domain object, created using the command layer (Figure 7).

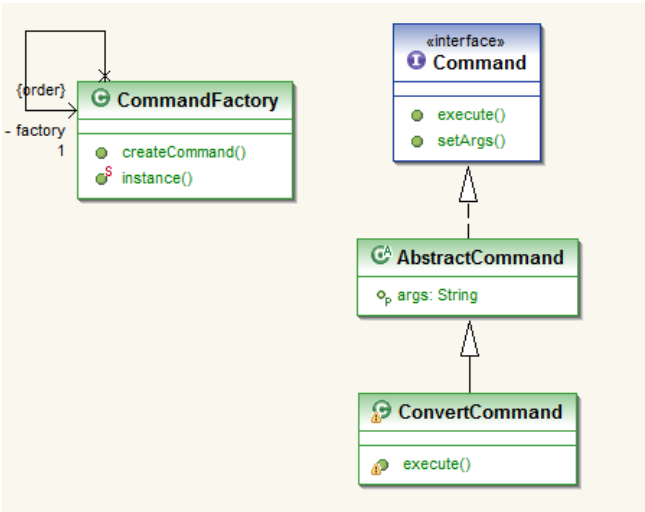

Figure 8: The command factory, a static singleton, is accessed using the instance function. The createCommand method in the CommandFactory creates a new command which is executed via the execute function.

Communication with the controller layer is established through the static factory singelton. The factory creates the appropriate command based on the input and executes it. Each command must implement the command interface in order to be compatible. This interface has two functions;

execute and setArgs. The execute function executes the command and returns a DNML domain object. The setArgs method allows the customization of a command. The factory includes a property file to find command location by command name at runtime. This structure is very important for extensibility and modularity. Hence, when a new command needs to be added, there is no need to rebuild the library. The control layer submits the task to the service layer after the process is started by command (Figure 8).

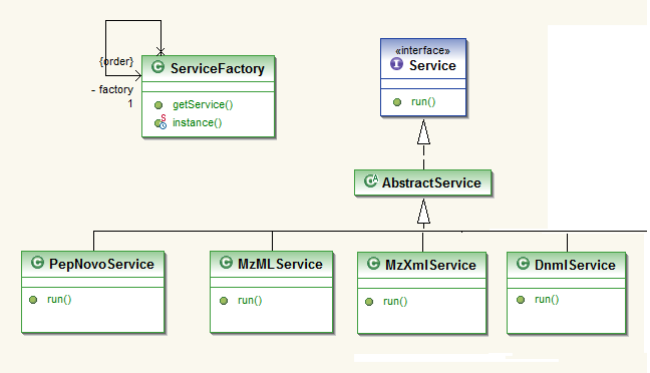

Figure 9: The service package is accessed via the instance method in the ServiceFactory. A new service is created by the getService method which is then executed by the run method.

In order to ensure modularity of the service layer, all services are created independently. As with the previously mentioned layers, the service layer factory is a static singleton. The factory creates suitable service depending on the given input and in collaboration with the service layer ensures their execution. Each service must implement the services interface which has an object container for handling input as a string array. The first operation is parsing the input using regular expressions and to check whether the input is suitable in respect to security and conformity to the service factory's expectations. When successful, the requested service is run employing the given input. The output is again modelled as an object container which is important for the concept of modularity. Thus, it helps us to provide great flexibility for future extensions (Figure 9).
This flexibility is also implemented on the mapping between XML and parsing classes. The domain layer (not shown) contains classes which, depending on XJB for JAXB, seamlessly propagate changes between the DNML schema and the parser. This technology creates flexibility allowing us to quickly respond to new developments in the field which may entail changes to the DNML schema.

6. RESULTS

For the researcher it is of utmost importance to have access to data contained in any file format. Often we detected corruption while parsing files or found files that did not exactly adhere to their target standards. This is clearly undesirable and most likely due to the redundant development of APIs by different vendors. The API we provide removes this problem since all functionality, reading, creating, and writing are available thus removing the possibility of creating files that do not validate or are corrupted. This also enhances the confidence of developers of downstream tools, building on data provided in DNML format. The link we provide to spectra in mzXML and mzML format which can be accessed by id or can be extracted as text further enhances the ease with which our API can be used. Our API thus allows access to all necessary input for *de novo* sequencing and all output thus completely removing file handling from the developers of *de novo* algorithms. Since many spectra have been processed by existing *de novo* prediction tools and since those results are not available in DNML format we provide conversion facilities to the currently most prominent free tools, LutefiskXP and PepNovo but further conversion facilities for PEAKS and other tools are under way.
The DNML schema and the fully fledged API it comes with are the main outcomes of this study. We paid especial attention to extensibility of the API and most parts of the code allow for runtime binding of additional functionality thus making it possible for developers to easily extend our API. The API, which includes conversion functionality, can be downloaded from http://bioinformatics.iyte.edu.tr/dnml. The conversion functionality can also be directly accessed at http://bioinformatics.iyte.edu.tr/dnml/convert2DNML.php.

7. CONCLUSION

Currently, *de novo* sequencing, although not highly accurate, is gaining importance and interest . Unfortunately, much attention is diverted to unnecessary file handling which could be used to improve algorithms. We removed the input file handling problem by providing an API which allows developers to read spectra from the two most important standards, mzXML and mzML. We removed the output file handling problem by developing a standard file format, DNML, and providing an API to read, write, and create files in that standard. This will speed up development of future *de novo* sequencing algorithms and will also spawn the development of tools that build on *de novo* prediction like for instance the GenomicPeptideFinder .

8. OUTLOOK

With the development of the first version of DNML, finished, we now aim to enhance the API by including further ontologies in addition to the PTM ontology that we are currently using. We also aim to restrict possible values for most fields in the DNML standard by using a custom ontology. Independently from this we will provide further conversion functionality for more *de novo* sequencing tools. Our secondary aim is to develop an analysis and knowledge mining pipeline build on the DNML standard. But the immediate goal is a visualization module which allows the visual exploration of the data stored in DNML format.

9. ACKNOWLEGEMENTS

We would like to thank Prof. Dr. Anne Frary for proof reading. This study was in part supported by the Turkish Academy of Sciences.

10. REFERENCES

[1] R. Aebersold and M. Mann, "Mass spectrometry-based proteomics," *Nature*, vol. 422, no. 6928, pp. 198-207, Mar. 2003.

[2] E. A. Kapp et al., "An evaluation, comparison, and accurate benchmarking of several publicly available MS/MS search algorithms: sensitivity and specificity analysis," *Proteomics*, vol. 5, no. 13, pp. 3475-90, Aug. 2005.

[3] J. Allmer, C. H. Markert, E. J. Stauber, and M. Hippler, "A new approach that allows identification of intron-split peptides from mass spectrometric data in genomic databases.," *FEBS Letters*, vol. 562, no. 1-3, pp. 202-6, Mar. 2004.

[4] J. Allmer, "Algorithms for the *de novo* sequencing of peptides from tandem mass spectra.," *Expert review of proteomics*, vol. 8, no. 5, pp. 645-57, Oct. 2011.

[5] P. G. A. Pedrioli et al., "A common open representation of mass spectrometry data and its application to proteomics research.," *Nature biotechnology*, vol. 22, no. 11, pp. 1459-66, Nov. 2004.

[6] S. Orchard, C. Taylor, H. Hermjakob, W. Zhu, R. Julian, and R. Apweiler, "Current status of proteomic standards development.," *Expert review of proteomics*, vol. 1, no. 2, pp. 179-83, Aug. 2004.

[7] L. Martens et al., "mzML--a community standard for mass spectrometry data.," *Molecular & cellular proteomics•: MCP*, vol. 10, no. 1, p. R110.000133, Jan. 2011.

[8] A. Keller, J. Eng, N. Zhang, X.-jun Li, and R. Aebersold, "A uniform proteomics MS/MS analysis platform utilizing open XML file formats.," *Molecular systems biology*, vol. 1, p. 2005.0017, Jan. 2005.

[9] B. Ma et al., "PEAKS: powerful software for peptide *de novo* sequencing by tandem mass spectrometry.," *Rapid communications in mass spectrometry•: RCM*, vol. 17, no. 20, pp. 2337-42, Jan. 2003.

[10] A. Frank and P. Pevzner, "PepNovo: *de novo* peptide sequencing via probabilistic network modeling," *Anal Chem*, vol. 77, no. 4, pp. 964-73, Feb. 2005.

[11] J. A. Taylor and R. S. Johnson, "Implementation and Uses of Automated de Novo Peptide Sequencing by Tandem Mass Spectrometry," *Analytical Chemistry*, vol. 73, no. 11, pp. 2594-2604, Jun. 2001.

[12] D. K. Han, J. Eng, H. Zhou, and R. Aebersold, "Quantitative profiling of differentiation-induced microsomal proteins using isotope-coded affinity tags and mass spectrometry," *Nature Biotechnology*, vol. 19, no. 10, pp. 946-51, Oct. 2001.

[13] A. Keller, A. I. Nesvizhskii, E. Kolker, and R. Aebersold, "Empirical statistical model to estimate the accuracy of peptide identifications made by MS/MS and database search," *Anal Chem*, vol. 74, no. 20, pp. 5383-92, Oct. 2002.

[14] "Eclipse." [Online]. Available: http://www.eclipse.org/. [Accessed: 12-Jan-2012].

[15] "Origo | Origo." [Online]. Available: http://www.origo.ethz.ch/. [Accessed: 12-Jan-2012].

[16] "Soyatec - Open Solution Company: XAML for Java, UML for Eclipse and BPMN designer." [Online]. Available: http://www.soyatec.com/euml2/. [Accessed: 12-Jan-2012].

[17] "UML, BPMN and Database Tool for Software Development." [Online]. Available: http://www.visual-paradigm.com/. [Accessed: 12-Jan-2012].

[18] "sax parser." [Online]. Available: http://www.saxproject.org/. [Accessed: 12-Jan-2012].

[19] "JAXB Reference Implementation — Java.net." [Online]. Available: http://jaxb.java.net/. [Accessed: 12-Jan-2012].

[20] "Software:JRAP - SPCTools." [Online]. Available: http://tools.proteomecenter.org/wiki/index.php?title=Software:JRAP. [Accessed: 12-Jan-2012].

[21] "jmzreader." [Online]. Available: http://code.google.com/p/jmzreader/. [Accessed: 12-Jan-2012].