Encoding, Storing and Searching of Analytical Properties and Assigned Metabolite Structures

Inaugural-Dissertation

zur

Erlangung des Doktorgrades

der Mathematisch-Naturwissenschaftlichen Fakultät

der Universität zu Köln

vorgelegt von

Tobias Helmus

aus Münster

Köln, 2007

Berichterstatter:	PD Dr. C. Steinbeck Prof. Dr. D. Schomburg
Tag der mündlichen Prüfung:	11.06.2007

Abstract

Metabolites and other small organic molecules are of major importance in many different fields of natural sciences. They play crucial roles in metabolic networks, and knowledge about their properties and interactions helps to understand complex biological processes and whole biological systems. Thus, data describing small organic molecules on a structural level is recorded in a multitude of biological and chemical laboratories on a daily basis. Consequently, a large amount of highly interconnected data already exists and continuously is produced. This leads to a strong need for software systems and data formats supporting the scientists in exchanging, processing, storing and searching molecular data under preservation of its semantics.

The aim of this project was to develop tools, applications and algorithms to be used for the efficient encoding, collection, normalisation and analysis of this data. These should be supportive in the process of dereplication, structure elucidation, analysis of molecular interactions and publication of the so gained knowledge. It frequently is impossible, or at least very difficult and time consuming, to determine the structure and functionality of an unknown compound directly. Therefore, this commonly is realised indirectly by describing a molecule via its properties. In a next step, these properties can be used to predict its structural and functional features.

In this context, tools were developed, that allow the visualisation of structural and spectral data, the structured displaying and manipulation of extending meta data and properties as well as the import and export of a variety of spectroscopic and structural data formats. This functionality was extended by applications enabling the assignment of structural and spectroscopic features to each other and analysis methods. Additionally, a framework for the structured deposition and management of large amounts of molecular data in the file system and in various relational database systems was created. To ensure the lossless encoding of spectroscopic data under preservation of its semantics, an open, standardised and highly structured data specification was defined - *CMLSpect*. *CMLSpect* is extending the existing *CML* (*Chemical Markup Language*) vocabulary and therewith allows for easy handling of connected structural and spectroscopic information.

The set of applications and methods developed in the course of this project was integrated into the *Bioclipse* platform for bio- and chemoinformatics, providing the user with a high quality interface and developers with an easy to extend plug-in architecture.

Zusammenfassung

Informationen über Metabolite und andere kleine organische Moleküle sind von entscheidender Bedeutung in vielen verschiedenen Bereichen der Naturwissenschaften. Sie spielen z.B. eine entscheidende Rolle in metabolischen Netzwerken und das Wissen über ihre Eigenschaften, hilft komplexe biologische Prozesse und komplette biologische Systeme zu verstehen. Da in biologischen und chemischen Laboren täglich Daten anfallen, welche diese Moleküle beschreiben, existiert eine umfassende Datengrundlage, die sich kontinuierlich erweitert. Um Wissenschaftlern die Verarbeitung, den Austausch, die Archivierung und die Suche innerhalb dieser Informationen unter Erhaltung der semantischen Zusammenhänge zu ermöglichen, sind komplexe Softwaresysteme und Datenformate nötig.

Das Ziel dieses Projektes bestand darin, Anwendungen und Algorithmen zu entwickeln, welche für die effiziente Kodierung, Sammlung, Normalisierung und Analyse molekularer Daten genutzt werden können. Diese sollen Wissenschaftler bei der Strukturaufklärung, der Dereplikation, der Analyse von molekularen Wechselwirkungen und bei der Veröffentlichung des so gewonnenen Wissens unterstützen. Da die direkte Beschreibung der Struktur und der Funktionsweise einer unbekannten Verbindung sehr schwierig und aufwändig ist, wird dies hauptsächlich indirekt, mit Hilfe beschreibender Eigenschaften erreicht. Diese werden dann zur Vorhersage struktureller und funktioneller Charakteristika genutzt.

In diesem Zusammenhang wurden Programmmodule entwickelt, welche sowohl die Visualisierung von Struktur- und Spektroskopiedaten, die gegliederte Darstellung und Veränderung von Metadaten und Eigenschaften, als auch den Import und Export von verschiedenen Datenformaten erlauben. Diese wurden durch Methoden erweitert, welche es ermöglichen, die gewonnenen Informationen weitergehend zu analysieren und Struktur- und Spektroskopiedaten einander zuzuweisen. Außerdem wurde ein System zur strukturierten Archivierung und Verwaltung großer Mengen molekularer Daten und spektroskopischer Informationen, unter Beibehaltung der semantischen Zusammenhänge, sowohl im Dateisystem, als auch in Datenbanken, entwickelt. Um die verlustfreie Speicherung zu gewährleisten, wurde ein offenes und standardisiertes Datenformat definiert (*CMLSpect*). Dieses erweitert das existierende *CML* (*Chemical Markup Language*) Vokabular und erlaubt damit die einfache Handhabung von verknüpften Struktur- und

Spektroskopiedaten.

Die entwickelten Anwendungen wurden in das *Bioclipse* System für Bio- und Chemoinformatik eingebunden und bieten dem Nutzer damit eine hochqualitative Benutzeroberfläche und dem Entwickler eine leicht zu erweiternde modulare Programmarchitektur.

Abbreviations

ANDI	Analytical Data Interchange
AnIML	Analytical Information Markup Language
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASTM	American Society for Testing and Materials
AWT	Abstract Window Toolkit
BibTeXML	BibTeX Markup Language
BioML	Biopolymer Markup Language
BSML	Bioinformatic Sequence Markup Language
CAS	Chemical Abstract Service
CASE	Computer Assisted Structure Elucidation
CDK	Chemistry Development Kit
CI	Chemical Ionisation
CML	Chemical Markup Language
COSY	Correlation Spectroscopy
DOM	Document Object Model
DTD	Document Type Definitions
EBI	European Bioinformatics Institute
EI	Electron Impact Ionisation
ELN	Electronic Lab Notebook
EPL	Eclipse Public License
ESI	Electrospray Ionisation
FAB	Fast Atom Bombardment
FAQ	Frequently Asked Questions
GUI	Graphical User Interface
HMBC	Heteronuclear Multiple Bond Coherence
HQL	Hibernate Query Language
HSQC	Heteronuclear Single Quantum Coherence
HSQL	Hypersonic SQL Database
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
InChI	IUPAC International Chemical Identifier
IR	Infrared
IUPAC	International Union of Pure and Applied Chemistry
J2EE	Java 2 Platform, Enterprise Edition
JCAMP	Joint Committee on Atomic and Molecular Physical Data
JDBC	Java Database Connectivity
JFC	Java Foundation Classes
LAN	Local Area Network

LGPL	GNU Lesser General Public License
LIMS	Laboratory Information Management System
MALDI	Matrix Assisted Laser Desorption Ionisation
MathML	Mathematical Markup Language
MS	Mass Spectrometry
NIST	National Institute of Standard
NMR	Nuclear Magnetic Resonance
NOESY	Nuclear Overhauser Enhancement Spectroscopy
OLE	Object Linking and Embedding
OS	Operating System
OSGI	Open Services Gateway Initiative
OSI	Open Source Initiative
PCA	Principal Component Analysis
PDB	Protein Data Bank
PDBML	Protein Data Bank Markup Language
PNG	Portable Network Graphics
QSAR	Quantitative Structure-Activity Relationship
RCP	Rich Client Platform
RDBMS	Relational Database Management System
RDF	Resource Description Framework
RSS	Rich Site Summary, RDF Site Summary or Really Simple Syndication
SBML	Systems Biology Markup Language
SGML	Standard Generalized Markup Language
SMF	Service Management Framework
SMILES	Simplified Molecular Line Entry Specification
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
STMML	Scientific-Technical-Medical Markup Language
SVG	Scalable Vector Graphics
SWT	Standard Widget Toolkit
TCOSY	Total Correlation Spectroscopy
UI	User Interface
URI	Uniform Resource Identifier
UV	Ultraviolet
VM	Virtual Machine
W3C	World Wide Web Consortium
WAN	Wide Area Network
WSDL	Web Service Description Language
XHTML	Extensible HyperText Markup Language
XLink	XML Linking Language
XML	Extensible Markup Language
XPath	XML Path Language

XPointer	XML Pointer
XQuery	XML Query Language
XSD	XML Schema Definition
XSL	Extensible Stylesheet Language
XSL-FO	XSL Formatting Objects

Index of Tables

List of Figures

Table of Contents

1 Introduction

Systems biology aims at achieving a system level understanding of organisms and biological systems by integrating the data and information emerging from modern molecular biology. It uses data generated in e.g. *Genomics*, *Proteomics* and *Metabolomics*, to draw a complete picture of a system by analysing the interactions of the components and the resulting dynamics (see Chapter 1.1 for a more detailed introduction to systems biology).

These fields successively use the results of their predecessors to build new more complex perceptions with every step (shown in Figure 1). The identification of genomes enables the prediction of genes, leading to the possibility to predict their function. By integrating the function of a number of genes, it is possible to create pathways and connect them to metabolic networks. All this data is used in systems biology in an integrative way to model whole biological systems.

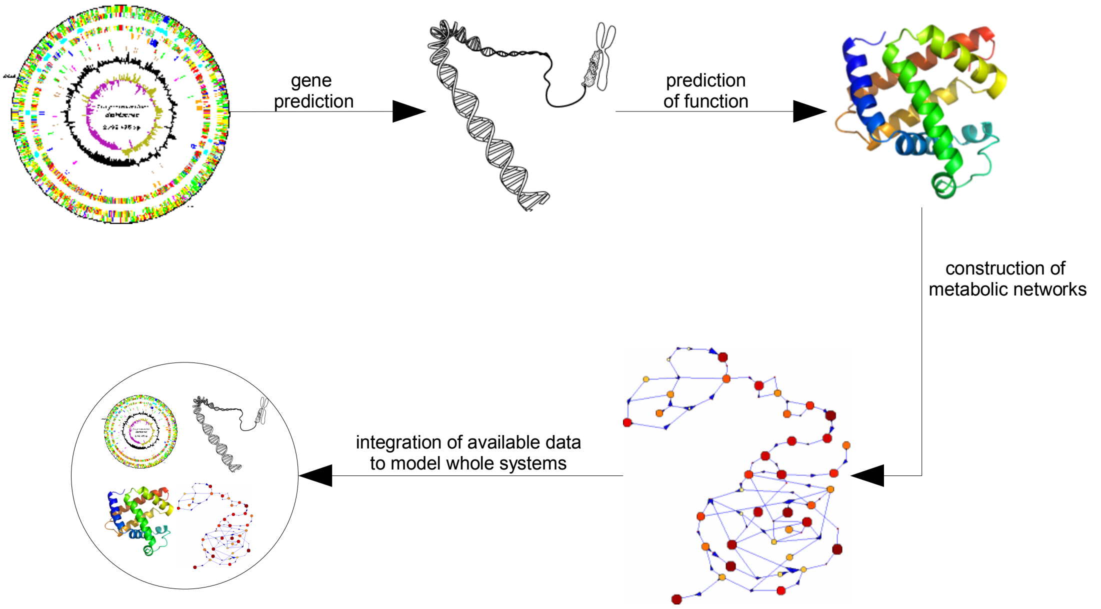

gene prediction
prediction of function
construction of metabolic networks
integration of available data to model whole systems

Figure 1: Diagram displaying the connection of the different "omics" fields and how the data generated by them is used by systems biology to understand complete organisms and systems.

The most recent of the "omics" fields, *Metabolomics,* focusses on the identification and quantification of an organisms metabolites, enzymes and their interactions. With this information large metabolic networks can be constructed, enabling the identification of metabolites and enzymes, which are crucial for the survival of an organism (see Chapter 1.1

page 7ff for more information on *Metabolomics*).

By abstracting these networks using graph theory, it is possible to determine parts of the network with special importance. Rahman et al. showed, that based on graph theoretical calculation of shortest path and connectivity information of metabolites, it is possible to identify the importance of single compounds. A "load point" in this model describes a highly connected hot spot in a metabolic network. "Choke points", in contrast, are forming bottlenecks within the network, as they describe enzymes, that uniquely consume or produce a certain metabolite [1] (see Figure 2).

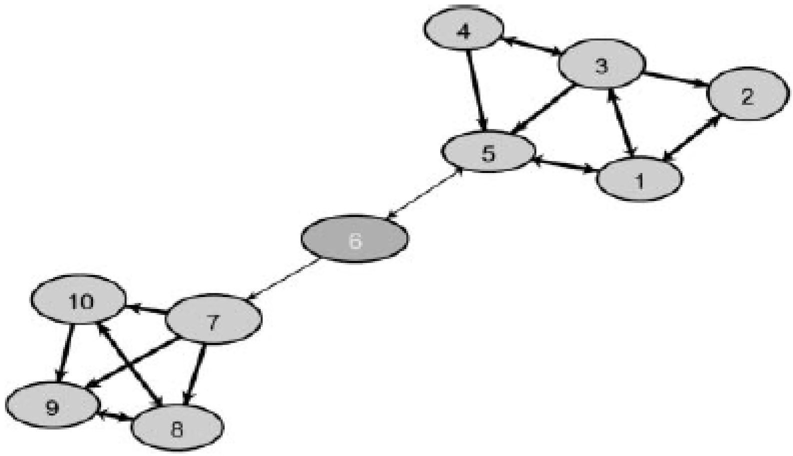


Figure 2: This depiction shows a section from a metabolic network represented as a graph. The nodes are representing the metabolites, whereas the edges stand for the enzymes and the reactions catalysed by them. The central grey coloured node (6) is a choke point as well as the adjacent edges. (Image taken from [1]).

An inactivation of a "choke point" would result in the regarding metabolite not being consumed or produced any more. This is of major interest in drug development, as these points present potential drug targets. Additionally, a "choke point" analysis could help to identify potential adverse effects of a drug candidate by performing similarity or interaction analyses of the determined choke points and the drug candidate under analysis.

As metabolites are small organic molecules and their interactions are based on chemical reactions, chemical information is needed to understand the single steps within such a network. For that reason, small molecules and the knowledge about their structure, their properties and their interactions are of major importance for reproducing and understanding metabolic networks.

This importance of small molecules leads to an increased demand for experimental information describing chemical compounds on a structural level. As the type of data used in this context (spectra, reactions, etc.) is generated on a daily basis in biological and chemical laboratories all over the world, one would expect a very solid knowledge base to be available to the scientific community. However, there exist several hurdles avoiding that this data is being re-used. These hurdles can be divided into three distinct classes:

- **Publications:** Information in science was traditionally published in printed media. Even thought this process changed towards digital publication, the related data is in most cases not stored with or linked to the publication, the information is published without its semantic and ontological context and the information is very difficult to be searched and extracted.
- **Storage:** A good deal of the data generated in the daily work of scientists is not stored in structured and communally agreed databases at all, or at least not freely accessible for machine processing.
- **Encoding:** Another major issue within scientific information management is the encoding of the available data. There are many different, very often proprietary data formats used for storing and exchanging data. This leads to information loss on conversion and makes it very difficult to process the encoded data in an automated manner by computer software.

A potential information flow for analytical data in life sciences is shown in Figure 3. This data is typically processed with the help of machine vendor applications and therefore is normally digitally available, but most often not in a format allowing for easy interchange. There is a strong need for communally agreed, openly specified and standardised data formats, enabling for easy processing, exchange and extension of the data. This would enable the data to be published and stored in some pool of freely accessible repositories.

These repositories do not need to be centralised and unified, but probably better form a pool of interconnected institutional and governmental storage systems. If data formats are agreed upon and scientific information is exchanged with its semantics, there is even software imaginable, that would help the scientist in validation and subsequent preparation of this data for the publication either in a scientific journal, a web page or anywhere else.

The current publication process of analytical data is erroneous, as the data itself normally is just partly published in form of analytical blocks within chemical publications (see Figure 42 for an example). To access this data by software systems, it is at the moment necessary to perform chemical archaeology by trying to recover as much of the information as possible. Therefore, an additional software layer for retrieving the relevant data is needed. This data might then form the basis for new experiments leading to new perceptions.

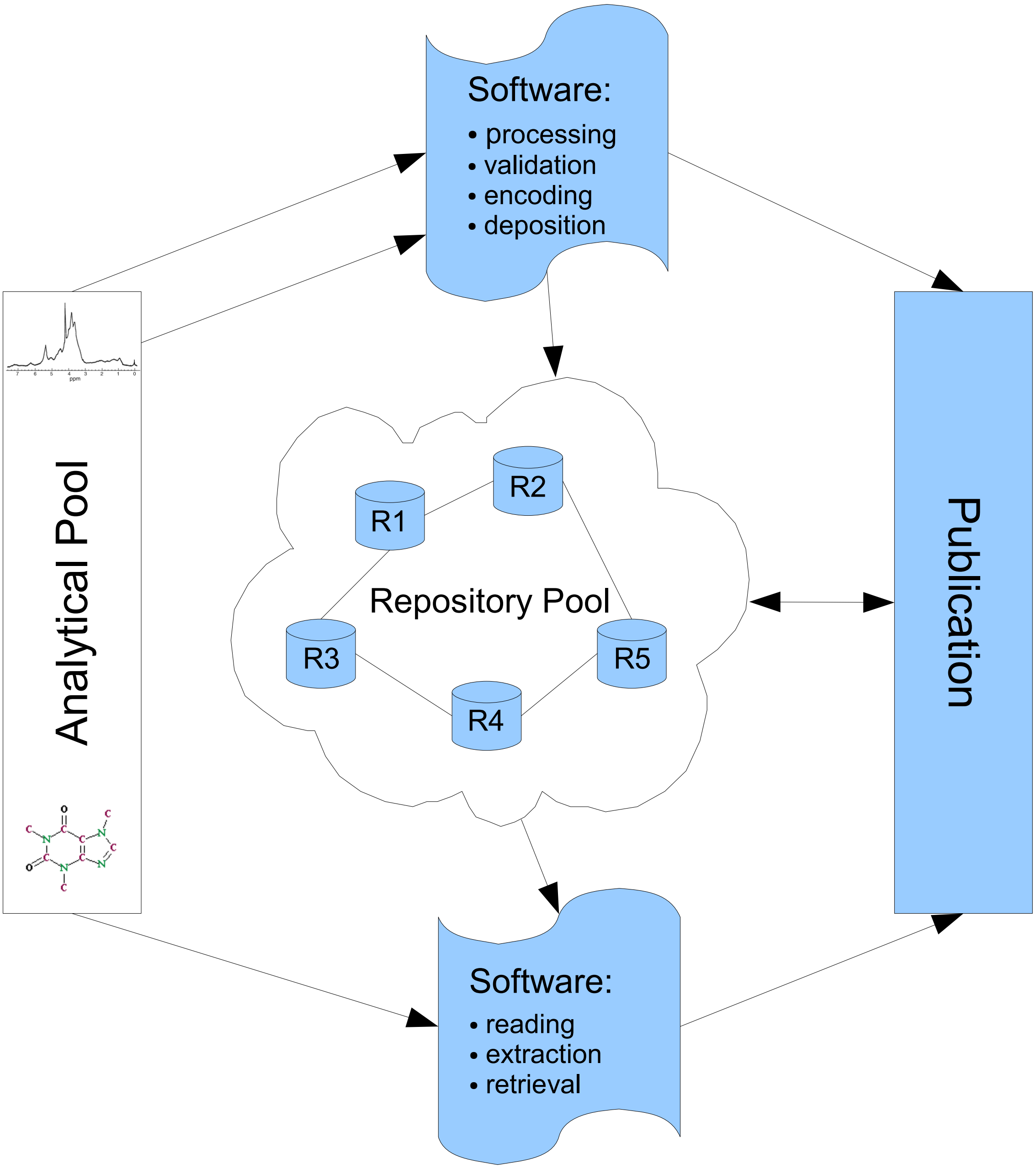


Figure 3: Schematic diagram showing a potential information flow within life sciences. Data is processed and validated by software systems, the extracted information is prepared for publication, whereas the data is being encoded and stored in data repositories. The direct linkage of the deposited data to the published information enables for computer based extraction of requested information and data.

Many of the tasks performed on chemical data are depending on large sets of data. Databases are e.g. used to check if a molecule under examination is already known. If this is not the case retrieval of additional information about the molecule or similar ones can be helpful in elucidating its structure and learn about its properties. In biology there already exists a number of open and freely accessible databases (e.g. *Protein Data Bank* (*PDB*) [2], *Kyoto Encyclopedia of Genes and Genomes* (*KEGG*) [3], *Universal Protein Resource* (*UniProt*) [4]). In chemistry in contrast, even though it has a long history of compiling data and storing it in large collections of information, these are in most cases only accessible for paying customers.

Nevertheless, there is a growing number of approaches towards this ideal flow of information, as there are more and more repositories being set up and many journals start to give authors the possibility to publish their scientific results in an open accessible manner. Furthermore, the first publishers start to enhance the traditional publication with data facts and semantics in machine processable formats [5].

Within this project, tools, algorithms, applications and data formats were developed, that support scientists in exchanging, processing, storing and searching molecular data under preservation of its semantics.

The other sections within this chapter give an introductory overview on general concepts that form the basis for this work and identify the necessity and demand this work is emerging from. The resulting overall objectives will be explained briefly in Chapter 2, followed by a presentation of the software systems used as a basis for the developed methods, algorithms and applications in Chapters 3 and 4. In Chapter 5 these applications, algorithms and methods and their underlying concepts, that were used and developed within this thesis, are described in detail. The last chapter is giving a condensed recapitulation of the results of this work and gives an outlook on future and ongoing projects.

1.1 Systems Biology and Metabolomics

Modern systems biology aims at understanding physiology and disease from the level of molecular pathways, regulatory networks, cells, tissues, organs and ultimately the whole organism. More generally it is described as aiming at the system-level understanding of biological systems as a whole [6] [7].

This is achieved by the accumulation, integration and analysis of complex data from multiple experimental and theoretical sources using tools developed in highly interdisciplinary environments [8]. These environments are built by scientist from very different fields of natural sciences (e.g. *Molecular Biology*, *Proteomics*, *Genomics*, *Metabolomics*, *Informatics*, *Physics*, *Mathematics*, *Biochemistry* and *Chemistry*), bringing in their knowledge and methods.

This challenging task was enabled by the strong progress in technical methodology and the large amount of new data associated with this. Important mile stones to be named here are the complete identification of the human genome [9] and the major improvements achieved in the field of high-throughput-methods.

In addition to the recently generated data, there is a large magnitude of relevant information “hidden” in printed media like journal articles and books. To make this information accessible, techniques for information extraction and data mining are used and enhanced [10]. Furthermore, the data has to be encoded in standardised, robust and long lasting data formats. To ensure as well the high quality as the easy availability of all this data to the scientists, it has to be shared in online databases and/or open data repositories [11].

Focussing on the whole set of components and their interactions within a system is in contrast to the “traditional” way of hypothesis driven science. There, it is common to break down a problem into smaller units, to examine these units separately and afterwards try to generalise the obtained conclusions by recombining them. However, this procedure is contrary to the general assembly of complex systems.

A complex system is built by interacting parts, whose interactions lead to new properties and functions. This formation of emergent properties causes an irreducibility of these systems, as they could never be monitored looking at their subunits alone. Complex systems are simulated on a hypothetical level by using modelling techniques from the field of computer sciences and mathematics. These models allow scientists to accomplish two very important tasks [10]:

- The prediction of a systems behaviour in reaction to any perturbation
- The redesign of a network to create new emergent system properties

For ensuring the lossless exchange of the so designed models and the related data, new formats are being developed that unify their expression (e.g. the *Systems Biology Markup Language* (*SBML*) [12]).

Thus, the scope of systems biology is ranging from data generation and integration, over computer based simulation of networks and systems, to experimental techniques like *in-vivo* modelling, by introducing perturbations into example organisms on different levels (genetic or environmental) [13].

A very large portion of the data used in system biology is derived from experiments within the so called "omics" fields, e.g. *Genomics*, *Transcriptomics*, *Proteomics* and *Metabolomics*. The tools developed within this work are mostly to be used for data handling of *Metabolomics* and other chemical data, but do not directly support data emerging from the other "omics" fields. Therefore, just *Metabolomics* will be described in detail in the following.

Metabolomics is the study of the whole metabolome of a biological system by identification and quantification of the contained metabolites and their relationships [14]. A metabolome is defined as being the "complement of metabolites of an organism" [15]. Metabolites are the intermediates and products of the metabolism, through which cells acquire energy and build cellular components.

Metabolic reactions can be divided into two different types [7]:

1. *Catabolic reactions:* gain of energy by breaking down complex compounds to smaller units
2. *Anabolic reactions:* consumption of energy for the construction of complex compounds

Metabolite examination is very commonly used in analytical biochemistry with a broad variety of methods applied for metabolite identification. The currently most frequently utilised procedures are [14][15]:

- The combination of a separation step (commonly gas or liquid chromatography) with mass spectrometry

- Analyses based on *Nuclear Magnetic Resonance* (*NMR*) spectroscopy

NMR spectroscopy has the advantage, that it is non destructive and therefore can better be used for the continuous measuring of metabolic profiles.

The analytical methods used to study the metabolome typically result in a large amount of high-dimensional data sets. For the interpretation of this data multivariate analysis methods like *Principal Component Analysis* (*PCA*), hierarchical clustering, evolutionary computing algorithms and other machine learning approaches are used [16].

One very common way of metabolite identification is the comparison of a pattern as unique as possible, describing the studied substance, with an existing library having reference patterns stored. This requires the existence of comprehensive and accessible data repositories with data stored in a standardised format. The qualitatively and sometimes even quantitatively determined data resulting from these interpretation steps is then used to infer biochemical networks or pathways. Beside the concentration of the molecules, their rates of change are of special interest for the modelling of metabolic networks.

Beside the academic interest of drawing a complete map of the constituents of a cell and their interactions, *Metabolomics* has an impact on a variety of applications, especially in medicine [16]. The investigation of the metabolome rises the probability of finding potential new drugs and/or new lead structures for drug development. By understanding metabolic pathways and networks the chance to intervene and to find ways to possibly increase or decrease the production rate of a certain metabolite grows. Metabolic profiles of a cell can be used for diagnosis of diseases, differentiation of healthy and diseased cells and can be used as quick tests for certain metabolic malfunctions [16][17].

Nobeli and Thornton highlighted in this context the importance of well organised and standardised public domain databases for the collection and retrieval of *Metabolomics* data and the development of open source software for data handling and data analysis especially in the field of chemoinformatics [16].

This is what the *Bioclipse* framework (see Chapter 4) is aimed at. It is providing supportive tools and applications for the different fields of systems biology. The applications and methods developed in this work are aimed at supporting scientists in the collection, administration and analysis of experimental data. The main focus is on data derived by metabolomic research whereas modules developed by other scientists cover the fields of proteomics and genomics research.

1.2 Computer Assisted Structure Elucidation

As scientists in such diverse fields like biochemistry, biotechnology, molecular biology, pharmacology and chemistry are constantly synthesizing new compounds or have the necessity of identifying newly discovered substances, the characterisation and structural elucidation of these compounds is of major importance.

Modern experimental techniques are generating data on a much higher rate, than it can be interpreted even by experts. Therefore, methods were and still are developed, that make use of computers within the process of structure determination.

In this context, chemoinformatics has long been developing tools for the *Computer Assisted Structure Elucidation* (*CASE*) of these unknown compounds. At the moment this field experiences a renaissance due to increased computer power, decreased memory prices and the decreased execution time of wet-lab experiments [18][19].

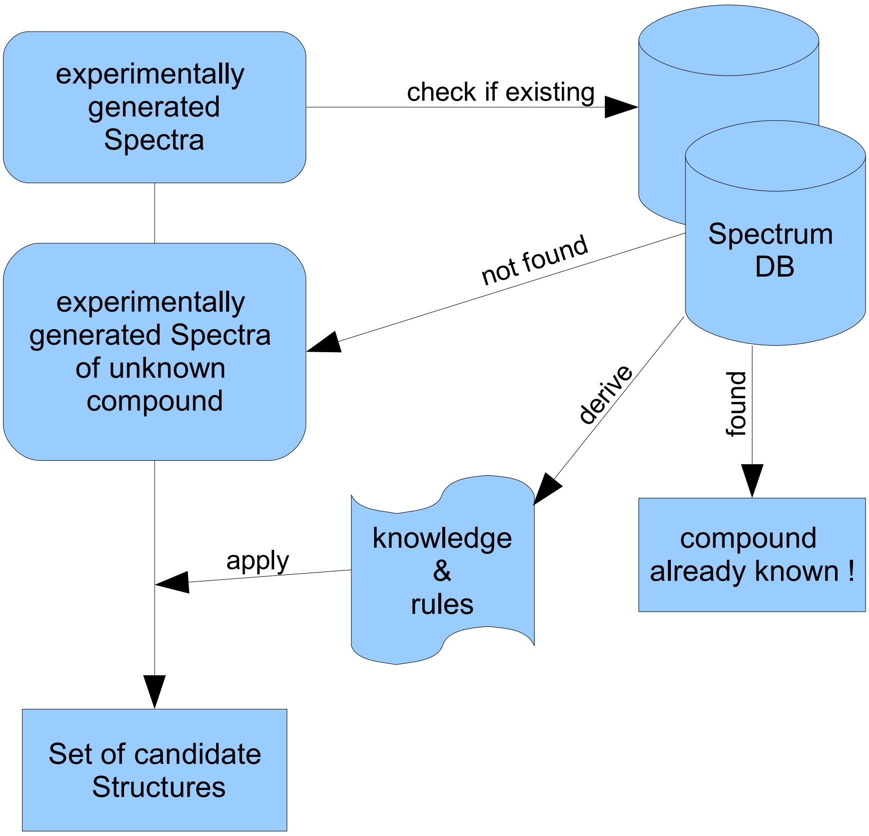

experimentally generated Spectra
check if existing
Spectrum DB
not found
experimentally generated Spectra of unknown compound
derive
found
knowledge & rules
compound already known !
apply
Set of candidate Structures

Figure 4: Schematic model of a dereplication process with attached Computer Assisted Structure Elucidation (CASE) steps.

CASE systems are typically knowledge and rule based systems, which derive structure information from spectroscopic data and use structure generators to build all possible isomers in agreement with the spectroscopic data (see Figure 4). Finally, these isomers will be verified by e.g. a comparison of predicted spectra with the original experimental data. The better the coverage with spectroscopic data, the fewer solution structures are suggested by these systems.

There exist mainly two different approaches to solve the structure generation process: deterministic and stochastic procedures. The deterministic methods try to generate all feasible structures, that match the input data. In contrast, the stochastic methods use algorithms and so called machine learning methods, that stochastically optimise the molecular structure towards agreement with given structural properties. Deterministic procedures are very often improved by the simulation of experts decisions and therefore are normally based on large knowledge bases. Furthermore, there exist hybrid approaches that combine these two procedures.

In order to derive rules or train machine learning methods, *CASE* systems often work on top of large databases of spectral data associated with structural features and physico-chemical properties of the molecule [20][21]. These databases are e.g. used for performing sub-spectrum and sub-structure searches to find good starting structures for the elucidation process. On the next level of this process, known correlations between spectra and structures are used to predict structural and/or spectral properties of compounds, that are not part of the database [22]. The last step in this process is the generation of fitting structures and the ranking of the so gained datasets [23][24][20].

Especially the different types of *Nuclear Magnetic Resonance* (*NMR*) spectroscopy (e.g. ^{13}*C-NMR* and the two dimensional techniques) are used for describing the properties of a structure in this context, but information from mass spectrometry and infrared spectroscopy is used as well.

There exist different software systems, that focus on sub-sets of these experimentally generated data, but there is an evolution visible towards methods combining all information available for a target [25].

Another very important step in the process of structure determination is the prediction of different spectra for probable target structures and the comparison of this artificial data with the existing experimental data [26] [27]. This is also used for the quality assurance of data to be included into existing spectroscopic database systems. By comparing the predicted and the measured spectrum it is possible to mark patterns as possibly erroneous if they diverge to much.

To avoid unnecessary timely and exhaustive *ab initio* structure elucidation, a pre- screening called dereplication is performed to exclude the possibility, that the compound under examination is already known (see Figure 39). This can be accomplished by executing spectral similarity searches on in-house or public structure-spectrum databases. Only if this search is unsuccessful, it is reasonable to reach for one of the more sophisticated *ab initio* tools for computer assisted structure elucidation [21].

1.3 Spectroscopy and Spectroscopic Data Formats

Methods summarised under the term “spectroscopy” are of major importance in applied chemistry, molecular biology, metabolic research and especially within the process of *Computer Assisted Structure Elucidation* (*CASE*).

Spectroscopy is a group of experimental procedures used to analyse the absorption or emission of energy of a studied substance in form of photons or electromagnetic waves. The energy difference of two quantum-mechanical states thereby is equivalent to the energy of a photon or respectively the frequency of an electromagnetic wave. This relationship is shown in the fundamental equation of spectroscopy:

$$\Delta E = h \cdot v$$

$$\begin{aligned} &\Delta E = the\ energy\ difference \\ &h = the\ Planck\ constant \\ &v = the\ frequency \end{aligned} \qquad (1)$$

The usual representation of spectroscopic data is the spectrum, a graphical display of a dimension proportional to the energy against the intensity.

As the difference in energy is dependent on the chemical composition of a substance, respectively the structure of a molecule, spectroscopic measurements are used by scientists to reveal information about quality and/or quantity of a certain assay.

Generally, the field of spectroscopy can be divided into three main types regarding their measuring process:

- *Absorption Spectroscopy:* the amount of light of a particular wavelength is measured, that is absorbed by a sample. (e.g. *IR Spectroscopy*, *UV/VIS Spectroscopy*)
- *Emission Spectroscopy:* the photon emission of a sample is measured. (e.g. *Fluorescence Spectroscopy*)
- *Scattering Spectroscopy:* measurement of the amount of light being scattered by a substance at certain wavelengths. (e.g. *Raman Spectroscopy*)

Chemistry uses spectroscopic methods to create fingerprints of molecules and to understand the chemical structure of a molecule and its properties. As some of the methods are realisable in an automatic and semi-automatic way, they can be used in so called high throughput assays as well.

The most commonly used methods in this context are infrared (*IR)* and *Nuclear Magnetic Resonance* (*NMR*) spectroscopy and will be explained in the following in more detail.

As the name already reveals, infrared spectroscopy uses light in the infrared part of the electromagnetic spectrum. The part with greatest interest for organic chemistry is the one between 4000 – 500 cm^{-1}.

Molecules are no static constructs, but their atoms are constantly oscillating around average positions. This vibration leads to continuous changes in bond length and angles. If a molecule is exposed to infrared radiation, energy gets absorbed and the vibrational state of certain bonds changes. There are three types of molecular vibration, that lead to the different types of motion within a molecule:

- Stretching = change in the bond length
- Bending = change in the bond angle
- Torsion = for four atoms bonded together in a straight chain, the torsional angle is the angle between the plane formed by the first three atoms and the plane formed by the last three atoms.

The first two are of relevance for *IR* spectroscopy. Each of these two has several variations. Both can show symmetrical movements, meaning that two atoms show the same directed movement, or asymmetric, if the atoms move antipodal. Additionally, the bending can lead to a movement within or outside the plane of that molecule.

Thus, different substructures of a molecule can be determined by their characteristic pattern of changes in absorbed energy in dependency to certain frequencies. Therefore, *IR* spectroscopy is in chemistry often used for the identification of functional groups. As the whole procedure depends on molecular asymmetry it just works with asymmetric molecules or symmetric molecules showing asymmetric stretching or bending transitions [28].

In contrast, *Nuclear Magnetic Resonance* (*NMR*) spectroscopy is based on the spin of atomic nuclei and the interaction of these nuclei with their surrounding. Namely it is about the interaction of nuclei with each other, with magnetic fields, with the electron sheath of the atom and the electrons of the whole molecule.

By the information received from stimulating these nuclei with radio-frequency radiation it is possible to very accurately determine where certain atoms (primarily carbons and hydrogen's) are located within the molecule [28].

One dimensional *NMR* spectroscopy is used routinely by chemists for the determination of chemical structures (often in union with *IR* spectroscopy and mass spectrometry), for the detection of ingredients of a sample and for the examination of the interaction of molecules.

There exist several types of two dimensional *NMR* techniques as well. These include *Correlation Spectroscopy* (*COSY*), *Total Correlation Spectroscopy* (*TOCSY*), *Nuclear Overhauser Enhancement Spectroscopy* (*NOESY*), *Heteronuclear Single Quantum Coherence* experiments (*HSQC*) and *Heteronuclear Multiple Bond Coherence* experiments (*HMBC*). *2D-NMR* allows to visualise the couplings between different nuclei. This includes indirect spin-spin-couplings (via bonds) as well as direct spin-spin-couplings through space.

These techniques reveal more information about the studied molecule than one-dimensional experiments, as they provide information about the nature of the carbon backbone [28]. They are especially valuable to study molecules, that are of too complex structure to be easily and unambiguously determined with standard procedures.

Another technique, that is quite often being correlated with spectroscopy, is mass spectrometry. As the name already implies, this actually is no spectroscopic method, but used for similar analytical purposes.

In mass spectrometry organic or inorganic samples are first ionised, then separated by the mass to charge ratio of the created ions. Finally, they are registered by mass and abundance qualitatively and quantitatively. Samples can be ionised by different types of ionisation methods. The following list shows the most widely used methods:

- *Electron Impact Ionisation (EI):* By collision with electrons energy is transferred to the molecule. This leads to the creation of primary positively charged ions.

  $$M + e^{-} \rightarrow M^{+\cdot} + 2e^{-} \qquad (2)$$

  Occasionally, a two times positive charged molecular ion is resulting as well. These ions are quite unstable and therefore very often break down into smaller fragments. This fragmentation process is substance specific and reproducible.

- *Chemical Ionisation (CI):* An introduced gas is been ionised by *EI*. The generated ions react with the substance to be analysed and ionise it. The fragmentation rate is smaller than with *EI*.

- *Fast Atom Bombardment (FAB):* The analyte is bombarded with a particle beam of usually an inert gas like argon or xenon.
- *Electrospray Ionisation (ESI):* Chemical solutions of the analyte are atomised, ionised and the droplets then dried, so that just the ions of the analyte remain. This method is especially well suited for bigger molecules like e.g. proteins.
- *Matrix Assisted Laser Desorption Ionisation (MALDI):* For this method, the analyte is been fixated to a matrix and then co-crystallised. By bombardment of this crystal with a laser, particles are detached and ionised. This method is as well very suitable for the ionisation of larger molecules and often used for the ionisation of polymers and biopolymers.

A typical mass spectrometer consists of three different parts: an ion source, a mass analyser and a detector system. These components will be explained in further detail in the following using an *EI-Magnetic Sector Spectrometer* as example. Figure 5 shows a schematic depiction of such a system.

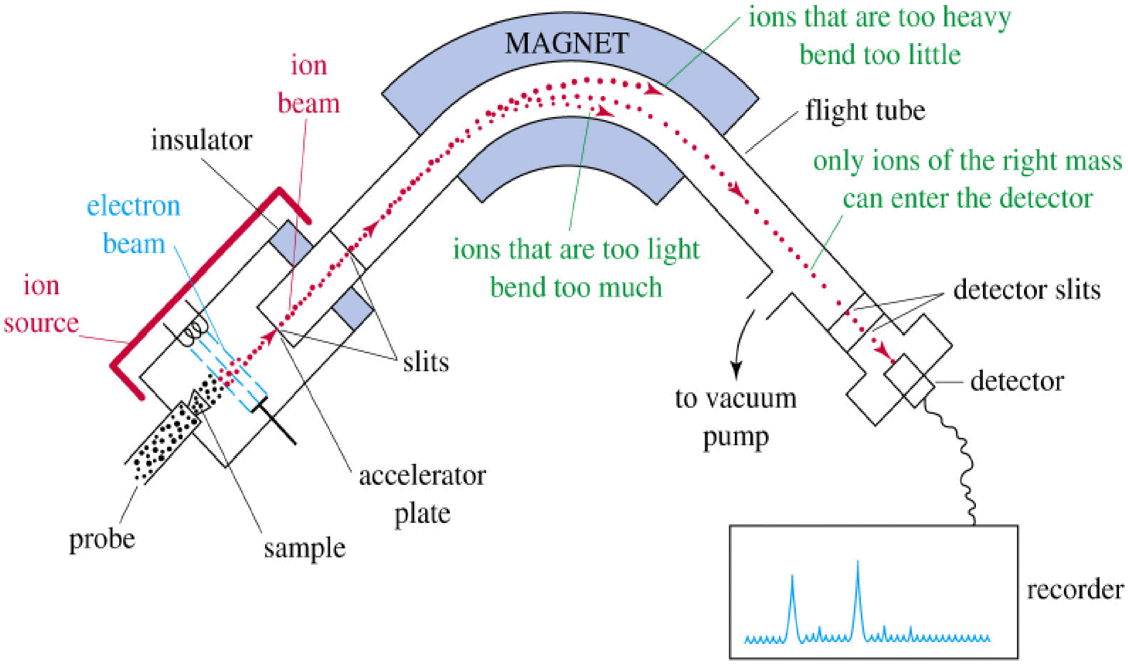


Figure 5: Schematic illustration of an *EI-Mass Spectrometer*. (image taken from Wikipedia - http://en.wikipedia.org/wiki/Image:Mass_spectrom.gif)

The analyte is introduced into the system, ionized by collision with electrons and accelerated into the mass analyser component of the device. The velocity of the ions is defined by:

$$v=\sqrt{\frac{2\cdot z\cdot U}{m}} \tag{3}$$

$$\begin{aligned} m &= ionic\ mass \\ z &= ionic\ charge \\ v &= velocity\ of\ the\ ions \\ U &= acceleration\ voltage \end{aligned}$$

Within the mass analyser the actual separation of the ionised fragments by their mass to charge ratio takes place. The ions go through a bent electric field and are deflected from their normal way of flight by this. The deflection radius is defined as follows:

$$r_m=\frac{m\cdot v}{z\cdot B} \tag{4}$$

$$B = magnetising\ force$$

By combining the two equations we get the fundamental mass spectroscopic equation:

$$\frac{m}{z}=\frac{{r_m}^2\cdot B^2}{2\cdot U} \tag{5}$$

Just if the ions are deflected on the right trajectory, they reach to the final component of the spectrometer, the detector. If not, they end upon the walls of the field block. In this system different masses can be separated and detected by varying the field strength of the magnetic field applied within the analyser part.

In comparison to the spectroscopic methods explained before, in *Mass Spectrometry* (*MS*) the mass to charge ratio instead of the energy is plotted against the intensity to create the final spectrum. There exist a series of different processes to perform the separation of the ions in the analyser part as well as different types of detecting components. Additionally, in modern devices there is very often an amplification step included for enhancing the resolution of the system before the detection .

In chemistry mass spectrometry is used for the identification of unknown structures, the definition of the molecular formula of an analyte, the quantification of a substance in a sample and for the determination of other physical, chemical or biological properties of compounds.

For very complex samples it is helpful to add a prior separation process before introducing them into the mass spectrometer. Therefore, *MS* is very often combined with gas- or liquid-chromatographic methods where the different ingredients of a mixture of probes are separated. These are very commonly used methods in protein and metabolite determination.

In all the mentioned spectroscopic and spectrometric procedures the use of automatic or semi-automatic methods for the data interpretation and data analysis is of growing importance, as they enhance quality and velocity of these steps [28].

1.3.1 Spectroscopic Data Formats

One of the basic requirements for a computer based analysis and/or interpretation of collected spectral information is its availability in a standardised machine readable format. Given this, informatics methods can be used for storing data in a structured manner in databases and to perform fast searches for substances and/or spectra against these databases. Additionally, they can be used for combing data gained by different methods or received from different databases to automatically or semi-automatically elucidate the structure of an unknown compound or to just validate a discovered substance.

There exist a variety of proprietary data formats used mostly by instrument vendors for the storage of spectral data measured with their systems. These formats cannot, or just in a limited way, be used for the exchange and long term archival of spectroscopic data. The reasons for this are difficulties in combining data from different systems, accessing the data with other software than the one it was recorded with and ensuring the readability of the data for the future. Therefore, there exist a number of open and standardised data formats, that try to overcome these problems. The following listing gives an overview on the data formats of most impact on the field of spectroscopy:

- *JCAMP-DX:* an open-source standardised file format for spectroscopic data developed and maintained by the *International Union of Pure and Applied Chemistry* (*IUPAC*). (see Chapter 5.2.2)

- *The Analytical Information Markup Language (AnIML): AnIML* is a web-aware mechanism for the instrument-to-instrument, application-to-application and instrument-to-application data exchange being developed by the *ASTM* (*American Society for Testing and Materials*) subcommittee E13.15. It is partly based on the *SpectroML Language* (*NIST – National Institute of Standard*) and the *Generalized Markup Language* (*Thermo Electron*). Additionally, it makes heavy use from older exchange formats like *JCAMP-DX* and *ANDI*. *AnIML* is using a layered approach to encode any type of analytical data [29].
- *The Analytical Data Interchange (ANDI) format: ANDI* is a standardised data interchange format mainly for mass spectrometry and chromatography developed by the *Analytical Instrumentation Association*. It tries to maintain the *GLP* (*Good Laboratory Practice*) and *GMP* (*Good Medicinal Practice*) integrity of the data [30].
- *Galactic SPC:* The file format used by all *Galactic* respectively *Thermo Galactic* products as exchange and storage format. Beginning with its invention this format was published in *Galactic's* documentation and via other public domain sources, a rare practice of *OEM* suppliers of instrument software. The format was designed to meet the needs of a user who wants to view, process and print the data outside the instrument vendor's software, but is not that well suited for data archival [31].

Another open, highly structured and machine processable data format was developed within this thesis in cooperation with the group of Dr. Murray Rust from the Unilever Centre for Molecular Informatics, Cambridge, UK. This is extending the *Chemical Markup Language* (see Chapter 6.2) by a vocabulary for spectral information named *CMLSpect* and is described in detail in Chapter 6.3.

1.4 Open Data, Open Source, Open Standard

As all software produced, all data generated and the formats defined within this thesis are made available in an open manner. The idea of open source, open data and open standard will shortly be explained in the following sections.

1.4.1 Open Source

Regarding the *Open Source Initiative* (*OSI*) open source software has to fulfil at least the following rules [32]:

- Its source code is either included, or is freely available
- The software can freely be copied, redistributed and used
- It can arbitrarily be modified
- The license has to be redistributed with the software and/or any derived software

The *OSI* is maintaining a list of licenses approved to fulfil this definition at http://opensource.org/licenses/.

Historically, in the beginning of software development all software was open. At that time software was normally distributed directly with the hardware and freely exchanged in user forums. In the 1980s the commercialisation of software started and the first open source/free software movements were born. In 1985 the *Free Software Foundation* was founded by Richard Stallmann supporting the free software movement. 1998 the *Open Source Movement* was floated establishing the *Open Source Initiative* and giving the already explained definition of open source software.

Eric S. Raymond, one of the co-founders of the *OSI*, summarised his view on software development and the advantages of open source software in an essay titled “The Cathedral and the Bazaar” first presented in 1997. This essay makes the case, that “*given enough eyeballs, all bugs are shallow"* – if the program sources are available for public testing, bugs will be discovered early [33].

In this essay Raymond opposes two software development models to each other. The “Cathedral” model, which is the typical model of proprietary software development, but of some open source projects as well, and the “Bazaar” like model adopted by e.g. the probably most known open source example - *Linux*. In this illustration the “Cathedral” is synonymous with the centralisation, slow release tempo, and vertical management of traditional software development, whereas “Bazaar” stands for code being developed in a collaborative approach, with many releases and a democratic management based on free accessibility of program and sources.

The ideas behind open source are very similar to the fundamentals of science. Both are based on producing, sharing, validating and, with the help of this process, improving information. This is manifested in the ongoing open access discussion leading to more and more freely accessible scientific publications and data [34]. Since scientific software is nothing else than a product resulting from scientific work, this discussion is inflicting the development of scientific software as well.

More and more scientific software projects evolve, that are based on open source development cycles. There were non-profit organisations, like the *Blue Obelisk Movement* [35] and the *OpenScience Project* [36], founded to support the development of freely available scientific software and to optimize interoperability between different scientific open source projects.

1.4.2 Open Standard & Open Data

To ensure long term access to data, two different requirements have to be fulfilled. First of all the accessibility of the data must be ensured. Much data at the moment, especially in chemical science, is stored in proprietary data repositories of commercial enterprises. In contrast to this, the term open data describes data, that is and will be, freely accessible for anyone. With the growing acceptance of open access more and more organisations are starting to set up their own data repositories, many of which store data in a way compliant to the open data ideas. This ensures the long term access to the data files themselves.

To furthermore make sure, that the information stored within these data files is assured as well, the data should be stored using openly standardised formats. An open format is a technical specification for digital encoding of specific data, usually maintained by a non-profit organisation, that is free of legal restriction on its usage. An example of an open format based data specification is the *Chemical Markup Language* (see Chapter 6.2).

1.5 Client-Server-Architecture

The client-server architecture is a basic concept of cooperative information processing where tasks are shared between programs on connected computers. The servers in such a system are offering some type of service, whereas the clients request these services on demand. The communication between servers and clients is normally transaction based, which means, that a client generates transactions (a sequence of logically connected actions) to be passed to the server for processing. Client and server might be connected within an application, via a local area network (*LAN*) or via a wide area network (*WAN*). It is not necessarily the case, that the computing power of the serving computer is exceeding that of the client; any combination of computing sources is imaginable and implementable. The main idea of the *Client-Server-Architecture* is the optimal use of the existing resources of all included systems. There are at the moment two major types of *Client-Server-Architectures* in use – *Thin Clients* and *Rich Clients* [37].

1.5.1 Thin Clients

A *Thin Client* is an application, that receives as much information from a connected server as possible, whereas the client is just responsible for the presentation of this data. In most cases the client as well handles the interaction with the user via any type of interface for creating the respective transactions to be submitted to the serving system. The server provides all the logic and computing power needed for processing the data and the generation of results. *Thin Clients* are widely used for browser based data handling, as the server can be written and compiled in any programming language, e.g. a *Java* internet server using *J2EE (Java 2 Platform, Enterprise Edition)*. The data transfer in these systems is realized via an existing intra- or the internet using *HTML* (*Hypertext Markup Language*) pages. A locally installed web browser is used for the presentation of the information and the interaction of users with the dialogue- and input elements. The advantages of a *Thin Client* architecture are:

- Easy to implement user interfaces
- The client side is normally completely platform independent
- The user can access information with the help of his/her favourite web browser → less training expenses

- easy or no installation on client side necessary

However, because *HTML* pages are static and there are just a limited number of dialogue elements available, it is difficult to produce high quality user friendly *Thin Client* applications. Therefore, often a lot of other techniques than *HTML* are embedded into the web pages as well. This leads to a lot of new advantages and drawbacks, but most often causes a platform dependency and less simple interfaces, which additionally are new to the user as well. Another disadvantage is, that without network connection the whole system is not capable to work at all, because of lack of server accessibility. Last but not least, the bandwidth of the network connection limits the data transfer between client and server, and as the client is heavily depending on data being provided by the server, this can, especially for computationally demanding tasks, form a bottle neck in an application.

1.5.2 Rich Clients

The *Rich Client* is a variation of the *Fat Client*, that in turn is the complete opposite of a *Thin Client*. The data processing is solely done within the client application, including the algorithmic logic as well as the *Graphical User Interface* (*GUI*). The *Rich Client* can be seen as a derivative of the *Fat Client* providing “richer” user experience and solutions by being lighter weight and based on a component model. Mostly it is a framework, which is extendable via modules or plug-ins. Another major difference to the classical *Fat Client* is that it is easier distributable and update-/upgradeable. The *Rich Client* is normally characterised by a local data handling, synchronisation of local data with a remote server, rich supply with *GUI* elements and seamless integration into the working environment. So a *Rich Client* provides what you expect of a “normal” desktop application extended with a connection to one or multiple servers. The *Rich Client* technology represents a combination of the strengths of *Fat Client* and *Thin Client* technology: rich user experience, high scalability, platform independence and fairly easy deploy and update. One example of a “state of the art” *Rich Client* framework is the *Eclipse Rich Client Platform*.

2 Aim of the Project

There are large amounts of highly interconnected data generated in systems biology and *Metabolomics* laboratories every day. Access to this data is very valuable if not necessary in the further process of data generation and evaluation. This and the other facts mentioned in the last chapter lead to a strong need for freely available software systems, that support scientists in:

- Encoding the data in specified, highly structured and communally agreed data formats
- Processing data in preparation for publication under conservation of semantics
- Searching for distinct facts in the multiplicity of available and upcoming data sources
- The exchange of data between colleagues and the storage of the data in freely accessible data repositories

This project is aimed at the implementation of tools, algorithms and applications helping scientists in exactly these processes as well as the definition of necessary encoding standards for the data. The applications are intended to be used by experimental scientists for efficient collection, normalisation and analysis of data recorded in biological and chemical laboratories with the aim of being helpful, e.g. within the process of dereplication and structure elucidation.

Because most *ab initio* structure elucidation methods are based on the combination of spectroscopic with structural data, the tools to be developed are related to the manipulation of this data. The methods will enable the user to visualize, manipulate and analyse structural and spectral data and assign structural, chemical, physical, biological and other relevant information to the spectral data. Furthermore, the management of large amounts of data either within the file system, an integrated database management system or remote storage systems accessed via web-services is to be implemented.

A schematic depiction of the information flow planned to form the basis of the project is shown in Figure 6. The underlying idea is to combine structural information with analytical and other relevant data, normalise this data to a open, standardised and machine processable format (in this particular case the *Chemical Markup Language*) and provide an effective storage system to persist and re-access the data.

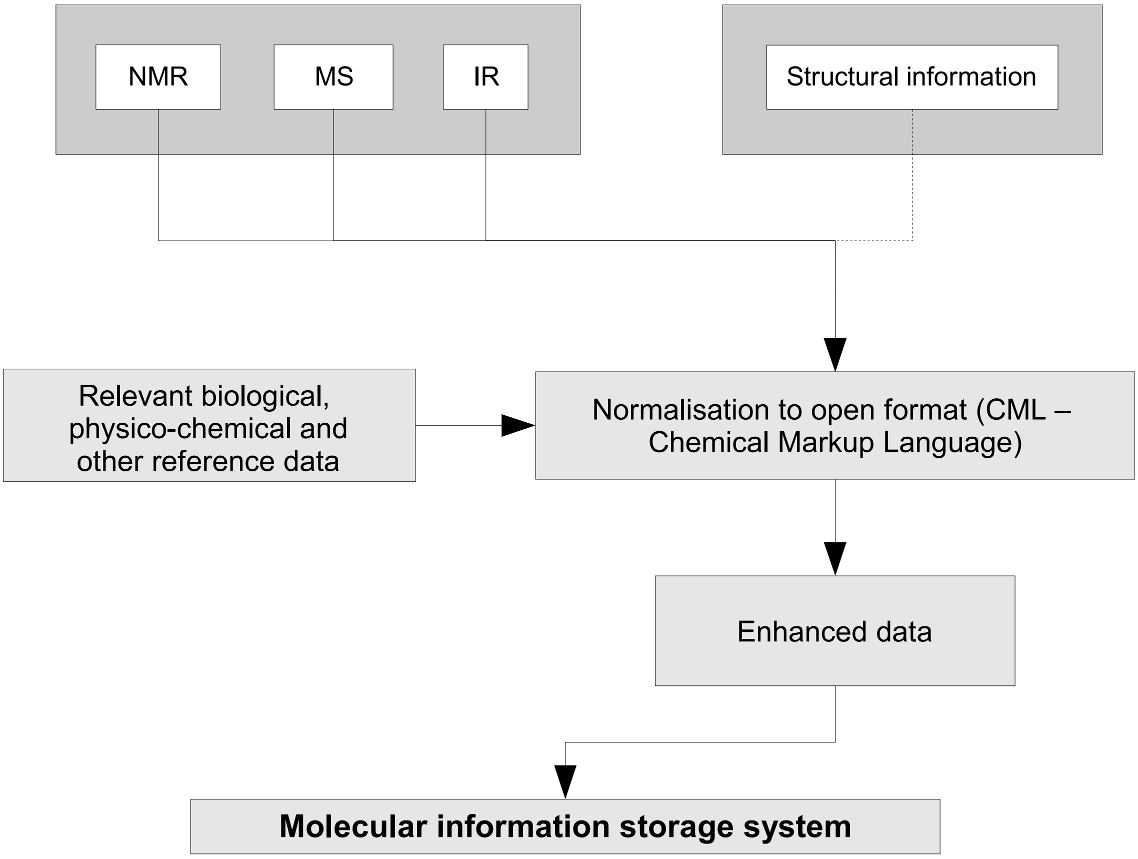

NMR
MS
IR
Structural information
Relevant biological, physico-chemical and other reference data
Normalisation to open format (CML – Chemical Markup Language)
Enhanced data
Molecular information storage system

Figure 6: Schematic diagram of the planned information flow within the tools and applications to be developed in the course of this project.

The decision to integrate the results of this work into the *Bioclipse* framework for bio- and chemoinformatics is based on the following facts (see Chapter 4 for more information on *Bioclipse*):

- *Bioclipse* is released under an open source license compatible to the existing licenses of our other projects.
- A mature and well designed code basis in the *Eclipse* project (see Chapter 3)
- An easy to adopt extension scheme
- The modular architecture of *Bioclipse* allows to extend the application without the need to apply major changes to the core of *Bioclipse* or *Eclipse.*
- *Bioclipse* is written in *Java* and therefore an easy integration of the already existing code is warranted.

All this leads to the anticipation of a broad acceptance on user as well as on developer side.

The outcome of this projects endeavours will be released under the terms of open source licences for different reasons. Beside the opinion that research funded by public organisations should be available to the public for free, the advantages of developing software as open source especially in an scientific environment should be mentioned. In my opinion this is one of the most productive ways of creating high quality software especially in an academic surrounding, as open source generally emphasizes quality and simplicity. This improves the project's chances to last for a longer period, even if they are not further maintained by their primary inventors.

3 Eclipse & Eclipse Rich Client Platform

Eclipse is an open source, independent platform managed by the *Eclipse Foundation*. It is described as "... A kind of universal tool platform — an open extensible *IDE* for anything and nothing in particular" [38].

Up to version 2.1 *Eclipse* was used as an *Integrated Development Environment* (*IDE*) only. Even for these early versions, the possibility to function as a basis for rich-client-applications as well, was considered. However, due to the fact that many *Eclipse* components were very strongly interwoven with the *Eclipse Workspace* and other *Eclipse* core components, this was impossible to realise at that point of time.

With the release of version 3.0 these problems were solved by a complete reorganisation of the platform resulting in an independent *Eclipse* core which forms the basis for any *Eclipse* based *Rich Client Platform (RCP)* application. The *Eclipse IDE* is in nowadays just another, but specific rich client application, build by the *Eclipse RCP* core extended by several different plug-ins [39] (see Figure 7).

Since version 3.1 the *Eclipse* core is formed by a *OSGi Server*, whereas the plug-ins are *OSGi* bundles. The *Open Service Gateway Initiative (OSGi)* is a worldwide consortium of technology innovators, that advances a proven and mature process to assure interoperability of applications and services based on its component integration platform.

The *OSGi* specifications define an in-virtual-machine *Service Oriented Architecture (SOA)* for networked systems. An *OSGi Service Platform* provides a standardized, component-oriented computing environment for cooperating networked services. This architecture significantly reduces the overall complexity of building, maintaining and deploying applications.

OSGi conform services can be executed on *OSGi* compliant servers like the *IBM SMF (Service Management Framework) Server* and *Suns Microsystems' Java Embedded Server*. *Java* applications are deployed to such a server as so called *Bundles*. One of the major advantages of using the *OSGi Service Platform* is that it provides functions to change the composition of the whole application dynamically at runtime [40][41].

3.1 Rich Client Platform

Most of the built-in functionality of the *Eclipse* platform is very generic. Therefore, it is necessary to extend the platform with additional tools, so that it can handle new content types, use existing content types in a different way and focus the generic functionality on specific requirements

The *Eclipse* platform provides developers with an elegant plug-in architecture, a native-looking user interface, and an easy-to-use help system. By utilizing a common framework for developing client-side applications, developers can focus their energies on addressing the specific requirements of their application instead of wasting time reinventing a set of core components.

The minimal set of plug-ins needed to build a rich client application is collectively known as *Eclipse Rich Client Platform* (*RCP*) [42]. The *RCP* is built upon the generic *w*orkbench including the *Standard Widget Toolkit* (*SWT* – see also Chapter 3.1.3.1), the *JFace* (see also Chapter 3.1.3.2) user-interface widgets and the *OSGi* runtime environment. These components provide developers a platform-independent *API* (*Application Programming Interface*), that is tightly integrated with the operating system's native windowing environment. In addition, it overcomes many of the implementation trade-off's that developers face when using the *Java Abstract Window Toolkit* (*AWT*) or *Java Foundation Classes* (*JFC*) [43].

The *RCP* is characterized by good interoperability with other technologies, being scalable from desktop computers to embedded devices, having a wide cross platform support including *Windows*, *Linux* and *Mac OS* and providing a high quality end user experience.

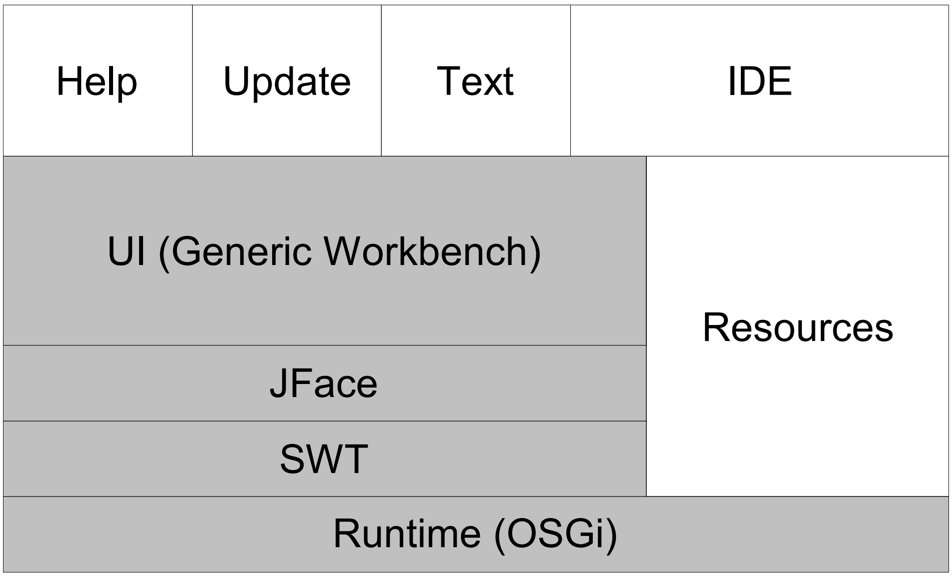


Figure 7: Schematic diagram of the components forming the *Eclipse IDE* as an example for any *Rich Client Platform* application. The grey parts are these components forming the core *RCP*.

The dependencies of the components forming the core of the *Rich Client Platform* and their connection to some optional components, which are very often used for building *RCP* based applications is displayed in Figure 7. The core itself is formed by the following elements:

- *Standard Widget Toolkit (SWT):* Provides developers a platform-independent *API* that is tightly integrated with the operating system's native windowing environment.
- *JFace Toolkit:* Platform-independent user interface *API* that extends and interoperates with *SWT*, includes a variety of components and utility classes.
- *Eclipse/OSGi Runtime:* Provides the foundation for plug-ins, extension points and extensions.
- *Generic Workbench:* Multi-window environment for managing views, editors, perspectives, actions, wizards, preference pages, etc.

The *Eclipse Platform* supports a variety of tools for application development, is not restricting the set of tool providers in any way and is supporting tools for the manipulation of arbitrary content types [44].

Regarding [44] "the *Eclipse Platform's* principal role is to provide tool providers with mechanisms to use, and rules to follow, that lead to seamlessly-integrated tools. These mechanisms are exposed via well-defined *API* interfaces, classes, and methods. The platform also provides useful building blocks and frameworks that facilitate developing new tools.”

3.1.1 Component Model

A software component is defined as a software entity providing a well defined function that can interact with other components via standardised interfaces to form a running program. Components are encapsulations of functionality and/or information that compose an independent unit of deployment and versioning [45].

Eclipse is based on the *OSGi* component framework, and therewith is implementing the defined component model as well. Indeed, the *Eclipse Platform* is composed by a server part (the *OSGi Server*) extended by a variable number of different *OSGi Bundles*, called plug-ins in the *Eclipse* environment. This component model is completely dynamic, so that any plug-in can be remotely installed, started, stopped, updated and uninstalled. A plug-in is the smallest and most essential unit within any *Eclipse RCP* application.

The interaction of the subunits in *Eclipse* is realised via extension points and extensions. An extension point is a mechanism whereby a new plug-in adds functionality to an existing plug-in or accesses the functionality of an existing plug-in using a defined extension.

Accordingly, every plug-in to plug-in interaction is defined by a pair of extension point (on the side of the plug-in being extended) and extension (on the side of the extending plug-in). The plug-in providing the extension point does not know anything about the extension. This leads to a strong encapsulation of functionality, which is of major help especially for implementing complex applications. A dynamic extension of objects in a plug-in is possible via adaptable interfaces [44]. A schematic depiction clarifying the plug-in interaction via extension points and extensions is shown in Figure 8.

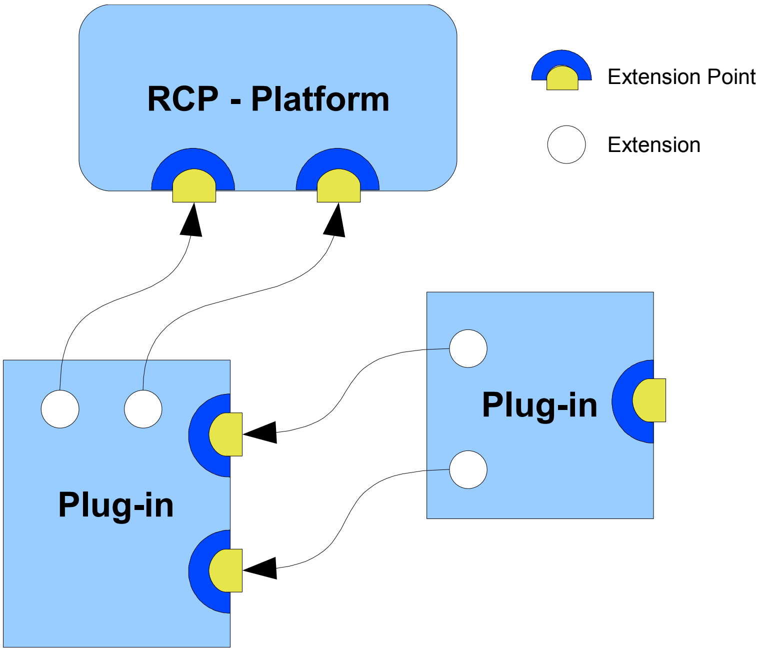


Figure 8: General schema showing the extension point - extension concept. The Eclipse Rich Client Platform can be extended by additional plug-ins via the provided extension points. Any plug-in itself can define new extension-points as well, that can themselves then be extended by other plug-ins.

On starting a *RCP* application the *Eclipse Platform Runtime* discovers which plug-ins are available and creates the plug-in registry - a listing of all registered plug-ins. This plug-in registry is a tree structure keyed by the plug-in identifier.

Although the platform registers all plug-ins, they are not loaded until first usage, a mechanism called *lazy loading*. This prevents the program from storing all plug-in related information in memory during runtime, which is especially useful for *RCP* applications consisting of many plug-ins.

At start up, only the initially required subset of plug-ins will be loaded. This reduces the amount of time and memory required for starting the application. One of the major advantages of the extension point concept is, that not even for collecting information from a plug-in via a declared extension point it is necessary to activate/load the whole plug-in [46].

Each plug-in in a *RCP* program declares its dependencies to other plug-ins and controls the visibility of its classes and libraries. The quality of the user experience depends significantly on how well the single tools integrate with the platform and how well the various tools interact.

Under normal circumstances a plug-in just has access to its internal classes and those imported from dependent plug-ins. Nevertheless, sometimes circular dependencies or other dependency problems avoid the direct accessibility of plug-ins. In this case the buddy concept, an extension to the general *Eclipse* class loading policy, can be used. By registering a component as a buddy to another one, it exposes itself to it and so enables the access to needed classes without explicitly importing them.

Beside the already mentioned lazy loading of components, the composition of *RCP* applications as a collection of separate plug-ins, which communicate to each other via the extension point mechanism, has several advantages. By enabling the individual upgradability of separate plug-ins, the amount of time and resources needed for updating the software is being decreased. By forming small units of functionality it is easy to reuse these in multiple contexts. Distributed development by a large developer community is a lot easier to organise, if every developer is responsible for a certain subset of components. Finally, the partitioning leads to a better overall design of the software project.

The distributable *RCP* application itself is created by assembling the various components to a coherent whole. This is realised via the product configuration, where, beside defining the plug-ins this product depends on, also the branding information (splash screen, window icons, about image and text, etc.) is provided [44].

3.1.2 Workspaces & Resources

The central place in *Eclipse*, where user data is managed is called workspace. This workspace consists of one or multiple top-level projects that map to a user-specified directory somewhere in the local file system. A project is a container for any number of folders and files. All objects within a workspace are called resources. The resources plug-in provides *APIs* for creating, navigating and manipulating these resources within a workspace.

Via the project's nature mechanism it is possible for a tool or plug-in to tag a project as a special project type, implying that the resources within this project are of a certain nature. For example, the *Java* nature marks a project containing the source code for a *Java* program. Plug-ins have the possibility to define new project natures and provide the code for the project configuration.

The workspace minimises the chance to accidentally loose information by keeping track of manipulations on resources within a low-level workspace history. Management of this history can be controlled by the user via preference settings.

There is also a marker mechanism implemented, that gives the possibility to supply a resource with annotations in form of meta data (e.g. to-do items, bookmarks, fix-me markings, etc.). Again this mechanism can be used and extended by extending plug-ins for storing individually defined types of markers.

Finally, there is a resource listener model implemented for keeping track of changes on workspace resources and for the distribution of this information to the different plug-ins registered to the application. Handling of the changes is done by storing a tree of resource deltas, mapping the entire process of resource creations, deletions and changes [44][47].

3.1.3 Workbench & UI Toolkits

The workbench is the top-level extensible *User Interface* (*UI*) element within *Eclipse*. It glues all *UI* components together and seamlessly integrates all tools by providing a common paradigm for the creation, management and navigation of workspace resources. Its *API* is built from two different toolkits:

- The *Standard Widget Toolkit* - *SWT*
- The *JFace* toolkit

3.1.3.1 The Standard Widget Toolkit – SWT

"*SWT* is an open source widget toolkit for *Java* designed to provide efficient, portable access to the user-interface facilities of the operating systems on which it is implemented" [48]. It is used throughout the whole *Eclipse* framework for the graphical interaction with the user. Wherever this is possible, it uses the operating systems' native widgets and just if not *SWT* provides a suitable emulation of that widget.

By using this approach, it stands somehow between the two approaches taken by the two widget toolkits distributed with *Sun Microsystems Java Platform: AWT (Abstract Window Toolkit)* and *Swing*. *AWT* just wraps *Java* code around native objects to create the *GUI* elements. To overcome *AWTs* shortcomings *Sun* introduced the *Swing* library.

Swing is completely *Java* based and emulates *GUI* widgets by calling low level *OS (Operating System)* routines to draw the *GUI* elements. It provides a lot more sophisticated elements than *AWT* and is entirely platform independent. By using the included look and feel mechanism *Swing* applications can look exactly the same even when running on different operation systems.

In contrast to this, *SWT* applications look and respond like native applications, and have tight integration with operating system features like the clipboard and drag-and-drop. Nevertheless, due to the use of native objects, which cannot be tracked by the *Java VM (Java Virtual Machine)*, these objects cannot be freed by the automatic garbage collection of the virtual machine, but have to be disposed manually by the programmer.

Additionally, because it wraps around platform specific libraries, *SWT* is not absolutely platform independent, but has to be adapted to any newly supported *OS*. Anyway, for all major systems (e.g., *MS Windows*, *Mac OS*, *Linux*, *FreeBSD*, and some more) there are separate *SWT* libraries available, so that distributing applications built on *SWT* should not become a problem.

One other drawback of *SWT* is, that there is no simple way of converting existing *Swing* based software to *SWT*. For handling this, a bridging mechanism has been implemented into *SWT*, that makes it possible to embed native *Swing* components into a *SWT* based *GUI* [42][44][49].

3.1.3.2 JFace

JFace is a *User Interface* (*UI)* toolkit, that is built on top of the raw widget system of the *Standard Widget Toolkit – SWT* without hiding it. It has several classes for many common *UI* programming tasks, strongly simplifying the overall user interface implementation.

SWT is in contrast to *Swing* not providing a controller-model-view object model in its controls and actions. This is done by *JFace* by providing so called viewers, model based adapters for certain *SWT* widgets like, e.g. lists, trees and tables. Configuration of this viewers is done via a content provider and a label provider. The label provider is delivering the specific label and icon for an element within the widget, whereas the content provider holds the correct mapping of input to the corresponding content.

Additionally to viewers, the *JFace* framework provides components for image and font registries, dialogs and wizards, actions and toolbar contributions and progress reporting for long running operations. The action mechanism gives the possibility to define actions independent from their exact location in the user interface. Each action holds its own *UI* properties, like the label, the icon, its tool tip, etc., and stands for a command that can be called via a menu item, toolbar item or button. Wizards and dialogs provide an easy way to compose a framework for complex interactions with the user.

All this gives developers the possibility to concentrate on the implementation of specific functionality, rather than on handling and arranging the underlying widgets [42][44].

3.1.3.3 Workbench

The term workbench in *Eclipse* is used synonymously with the *Eclipse Platform UI*. It provides the structure in which tools interact with the user within the *Rich Client Platform*. The workbench is built using both *SWT* and *JFace* and hence its *API* is depending on *SWT* and to a lesser extend on *JFace* as well.

The overall purpose of the *w*orkbench is to achieve seamless tool integration and controlled openness by providing a common paradigm for the creation, management, and navigation of workspace resources. This is realised by the definition of extension points within the workbench, that are extended by the tools to be plugged into the *UI*. The presentation and coordination of the user interface is then controlled by the workbench.

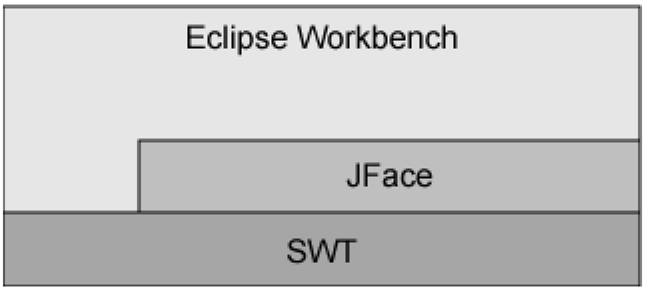


Figure 9: Relationship and interconnection of *SWT*, *JFace* and *Workbench*.

To the user, the workbench appears as a collection of buttons, menus, views, editors, dialogs and wizards. It is possible to define pre-set arrangements of these *GUI* elements in so called perspectives.

3.1.3.4 Perspectives

Each perspective provides a set of functionality aimed at accomplishing a specific task or working with specific types of resources. A perspective controls the initial view visibility, component layout, and action visibility in menus and toolbars.

Perspectives provide the possibility to rapidly switch from one working environment, for e.g. doing *Java* programming, to another, e.g. for the manipulation of *XML* files without having to separately open and arrange the windows needed. Nevertheless, the user has the possibility to customize any predefined perspective by just rearranging and/or resizing the defined components or by opening/adding new components to the actual *GUI* by using the related menus. The appearance of a the workbench is stored between working sessions, so that user introduced changes are not lost on closing the *RCP* application.

Most perspectives in a workbench are composed by one editor and one or multiple views. The *Eclipse Platform* comes with several standard perspectives for different purposes – e.g. online help, general resource navigation and team support tasks. Additional perspectives can be provided by enhancing plug-ins.

3.1.3.5 Editors & Views

Editors allow opening, editing and saving of an object the user is working on within the workspace. Every editor has its own load-save-close life cycle.

Activated editors can contribute actions to menus and toolbars. Contributions of editors, which are currently inactive, are “greyed out” and not selectable until the editor is re-activated. Different types of editors can be assigned to different types of resources or files.

If a resource, that has one or more editors assigned, is selected and opened a predefined preferred internal editor is used to display the resource's content. If no registered editors are found for the resource the workbench checks with the underlying operating system to determine, if it has any editors registered for the particular file type and tries to use that one to open the file.

It is possible to stack editors by using a *MultiPageEditor*, which is a collection of tabbed editors working on the same object.

Views support editors and provide alternative presentations as well as ways to navigate the information in the workbench. Views may provide their own menus and toolbars. The underlying actions should just affect the items within the view itself. The life cycle of views is a lot simpler than that of editors. Modifications, made within a certain view, are normally directly saved and reflected to other *UI* components (views/editors) working on this object. A view can occur separately or stacked with other views in tabbed form. Every view can be opened and closed separately, as well as it can be dragged and dropped to any other position within the workbench.

3.1.3.6 Wizards

Wizards are *GUI* elements, that guide users step by step through a set of operations by using a series of graphical dialogs.

There exist two different ways of integrating new wizards into the *Eclipse* system, either by using predefined extension points or by creating and launching the wizard on a programming level. If a plug-in uses an existing extension point for the creation of a wizard, the regarding actions to launch it are automatically generated and integrated by the workbench. The plug-in provider just needs to bring in the code for the wizard itself. Alternatively, the wizard can be started by an action, that has to be supplied by the plug-in developer as well.

The *JFace* classes used to build any type of wizard already form the general framework of the wizard itself. The developer just has to provide additional *GUI* elements and text, has to implement the sequence of tasks to be performed and has to ensure the final action to be executed past collecting all needed information.

3.1.4 Platform Integration

Any tools written in *Java* using *SWT* can be integrated seamlessly into the platform. If using *Swing* a quite high level of integration can be achieved by using the included "*Swing* to *SWT* bridge". However, even external tools can be launched from inside *Eclipse*. Although these must open in separate windows and can just access data with help of the underlying file system. This leads to a very tight integration into the system – especially on *UI* level. Depending on the operating system, *Eclipse* even supports embedding *OLE (Object Linking and Embedding)* documents and bridging to *ActiveX* elements.

3.1.5 Help System

By using the *Eclipse Platform* help mechanism any plug-in can contribute documentation to the application's global help pages. Therefore, the raw content of the help system is provided in form of *HTML* pages, whereas the structure of the pages has to be defined via *XML* files. This separation facilitates the integration of already existing *HTML*-documentation into the application.

The *Eclipse Help System* itself visualises these documentation pages by using the structuring *XML* files. It gives the user the possibility to browse, bookmark and print the documentation. Additionally, there is a full text search function letting the user search for phrases or keywords and a context-sensitive help, for finding information describing a particular function.

The interaction with the help system can occur in two ways, via an embedded help view, or by using a separate help window. Both are providing the same information, just in different ways.

3.1.6 Eclipse Summary

The *Eclipse Rich Client* platform is a very powerful framework for building rich client applications. It ensures good integration with the host environment, by providing a native look & feel, a sophisticated window management and being highly customisable using editors, views and wizards.

Because it is based on an *OSGi*-compliant component model, the system allows for dynamic component discovery and loading, as well as easy updating and extension. The simple to implement and extend help system reduces the time needed for potential users to be trained. All this reduces the time, costs and skill needed by developers to implement user friendly, rich applications.

4 The Bioclipse Framework

The *Bioclipse* project is aimed at creating a *Java* based, open source, visual platform for chemo- and bioinformatics based on the *Eclipse RCP* (*Rich Client Platform* – see Chapter 3). It is built on a plug-in architecture inheriting basic functionality and visual interfaces, like the help system, software updates, preferences, cross-platform deployment etc., from the *Eclipse RCP* system.

Bioclipse itself (respectively the plug-ins composing it) is providing functionality for chemo- and bioinformatics and extension points, that can easily be used for further extending the platform by additional plug-ins providing additional functionality [50].

Bioclipse inherits the platform-independence from the *Java* programming language. *Java* realises this, by being run in a virtual machine, which is freely available for all major operating systems (*Windows*, *Unix*, *Linux*, *Mac OS*). Because of this, it is not necessary to compile a *Java* application in dependence of the executing *Operating System* (*OS*).

This advantage is shortened a bit, as the graphical layer of *Eclipse* is bound to the underlying *OS* more tightly, than this is the case in other *Java* programs, not using the *SWT (Standard Widget Toolkit)* graphical toolkit for building the user interface. *RCP* applications have to be exported for the operating system they are expected to run on, but this still has no influence on the rest of the development process.

The real integration of several mature life science frameworks and components into a single framework, providing an intuitive user interface and a rich set of diverse functionality makes the *Bioclipse* framework the most advanced and user friendly open-source workbench for chemo- and bioinformatics [50].

The project is mainly developed as a collaboration between the *Dept. of Pharmaceutical Biosciences, Uppsala University, Sweden*, and the *Research Group for Molecular Informatics* at the *Cologne University Bioinformatics Center* (*CUBIC*), *Germany*. Beside these, there are further developers not belonging to one of these organisations contributing to the project as well. At the moment of writing there are 12 registered developers, of which 4-6 are contributing to the system on a daily basis.

Bioclipse is released under the *Eclipse Public License* (*EPL*), an open-source (see Chapter 1.4.1) software license, that is putting no constraints on the choice of back-end and/or license for creating plug-ins extending the system. Therewith, it is totally open for both, open source plug-ins as well as commercial ones.

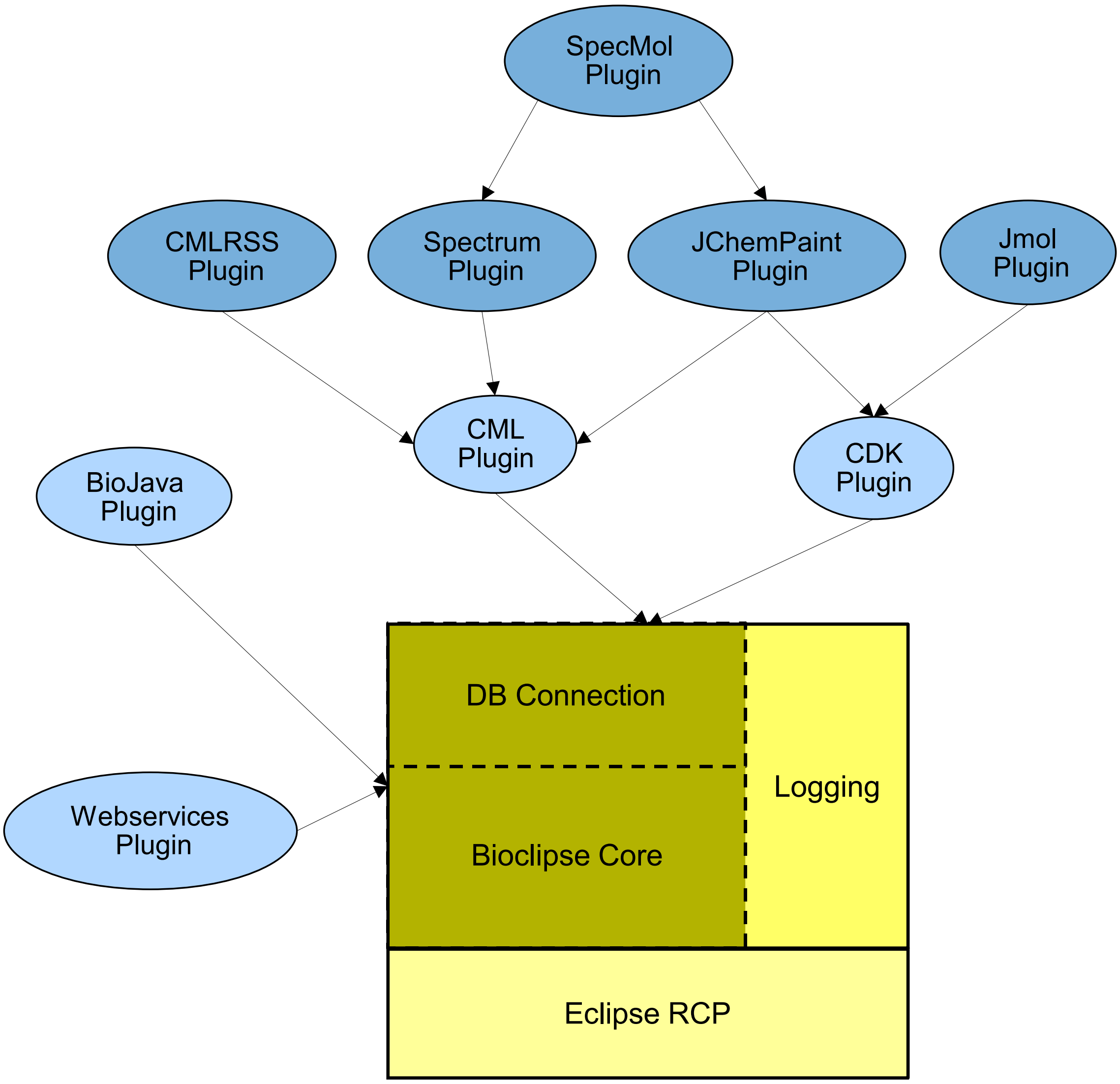


Figure 10: Diagram visualising the plug-ins forming the Bioclipse framework and their dependencies.

The *Eclipse* framework is used for constructing this universal tool platform specific for the fields of chemistry, biology, biotechnology, genomics, proteomics and pharmacology, altogether denoted as life sciences, by providing a wide variety of tools, algorithms and applications. All this functionality is distributed to a number of different plug-ins, making it possible for the user to select just the subset meeting the personal demands of functionality (see Figure 10).

To facilitate the usage and logical combination of the provided *UI* elements, three different perspectives were pre-defined. A perspective in an *Eclipse RCP* application is a collection of *UI* elements, which are grouped on a screen page and are all adding functionality to one topic (for further information on *Eclipse* perspectives see Chapter 3.1.3.4).

The following list shows the actually included plug-ins and gives a short description of their functionality:

- *CDK-plug-in:* Chemoinformatics back-end (see Chapter 5.1.1)
- *CML-plug-in:* Reading and writing *Chemical Markup Language* (*CML* – see Chapter 5.2.1)
- *Jmol plug-in:* 3D-visualisation of molecules and proteins
- *JChemPaint plug-in:* 2D-molecular editor (see Chapter 5.1.2)
- *SpecMol plug-in:* Functionality to handle, create and display so called *SpecMol* resources – *CML* based resources containing one molecule and one or multiple spectra related to this molecule (see Chapter 5.3)
- *NMRShiftDB plug-in:* Connects *Bioclipse* to the *NMRShiftDB* database
- *Webservices plug-in:* A framework to add web service functionality to *Bioclipse*
- *Database & DBResourceWrapper plug-ins:* Providing functionality to connect to *SQL* based database systems (see Chapter 5.4).
- *Spectrum plug-in:* Visualisation of spectral data (see Chapter 5.2.3)
- *CMLRSS plug-in: RSS*-viewer for retrieving chemical data in *CML* using *RSS* feeds
- *BioJava plug-in:* Analysis of sequences (*DNA*/*RNA*/protein) in various formats
- *Logging plug-in:* Logging capabilities using log4J

As already mentioned, *Bioclipse* can be prepared to fulfil different tasks by combining the plug-ins in different ways, so that sub-packages, e.g. for molecular chemistry are compilable and distributable. In the future, a web based service for checking out and updating of the individual plug-ins separately based on the *Eclipse* update mechanism is planned.

The *Bioclipse* core package provides some core functionality, that is needed for any *Bioclipse* based application. This plug-in is contributing the most general features to the framework. The core object model is defined here, with the *BioResource* implementing an interface called *IBioResource* as base object.

The object model is completely decoupled from the persistence implementation. This allows plug-in providers to extend the *BioResource* extension point for adding new resource types to the workbench without the need to care about the persistence of the new object (see Figure 11).

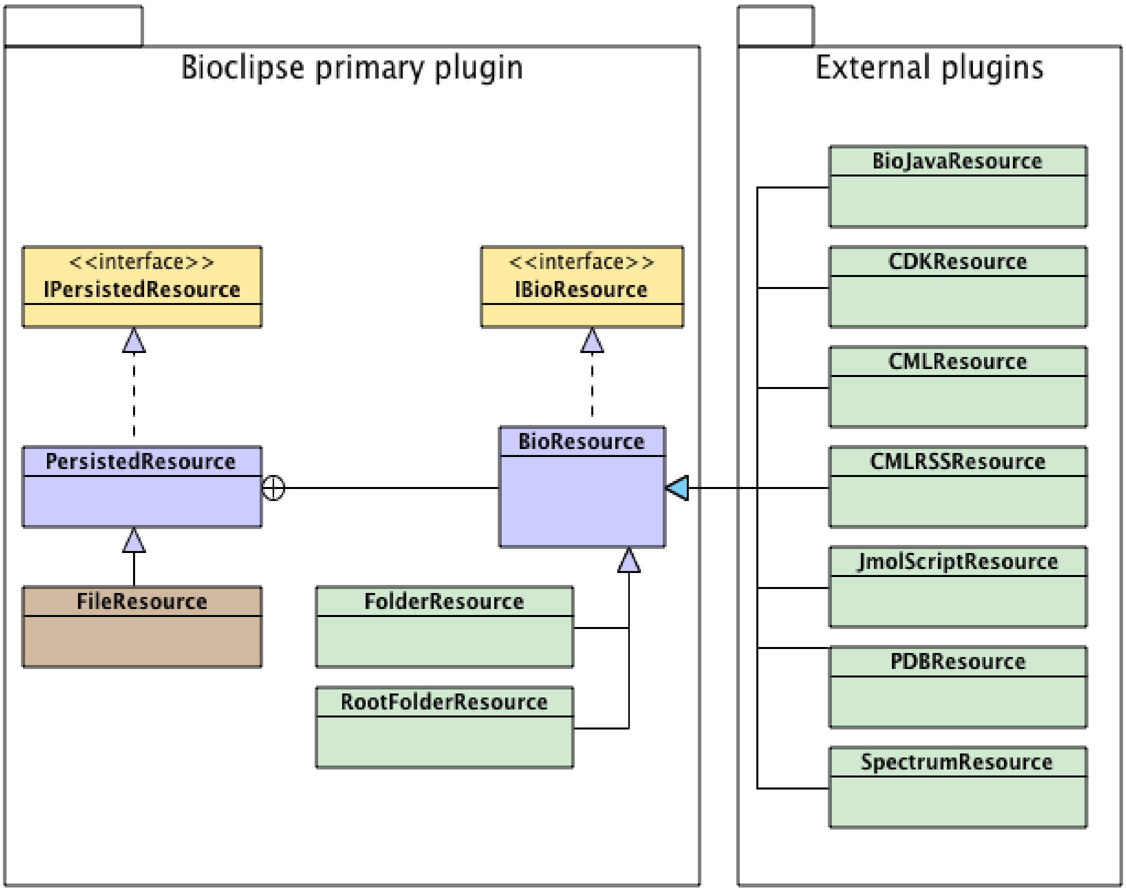


Figure 11: This class diagram shows the core object model of the *Bioclipse* platform and how additional objects integrate into this (image taken from [50]).

The central component in the platform is the *BioResource Navigator*, a tree-like view (see 3.1.3.5 for more information on *Eclipse* views) representing the *BioResource* hierarchy similar to the way the user is used to handle files and folders. It provides an extension point (see Chapter 3.1.1), that enables other plug-ins to contribute actions to the context menu of the resource tree, comes with wizards (see Chapter 3.1.3.6) for the creation of new basic resources (e.g. text based resources) and implementations for basic *UI* features like drag & drop and copy & paste.

Furthermore, two general editors (see 3.1.3.5 for further information on *Eclipse* editors), one for handling text files and another with extra functionality for handling *XML* context (e.g. syntax highlighting and code completion), are provided. Additionally, global actions like undo/redo and cut/paste, a view for displaying general properties of selected objects, a job scheduler for running timely exhaustive tasks in the background and a console displaying text messages to the user are added to the system.

Moreover, *Bioclipse* contains a searchable help system and an integrated preference system configuring and customising the workbench, both extendable via the respective extension points.

The bioinformatics functionality of *Bioclipse* is aggregated in the *Bioinformatics* perspective. This perspective is build by a collection of different views, editors and menus for loading, parsing, visualising, editing, converting and saving various formats of sequence and protein data.

For handling sequences the *BioJava* [51] library, an open-source framework for processing biological data, is used. This includes objects for manipulating biological sequences, file parsers, biological databases, and data analysis routines. With the help of a special view it is possible to display sequences together with assigned *SwissProt* features.

The 3D molecule viewer *Jmol* [52] is used for visualisation of macromolecules. The variant of this viewer embedded into *Bioclipse* is used for displaying 3D molecules within the chemoinformatics environment as well.

The web service plug-in of *Bioclipse* provides an exemplary general integration of web services into the framework. A web service is defined as a software system designed to support interoperable machine-to-machine interaction over a network. Web services as defined by the *W3C (World Wide Web Consortium)* use *SOAP* (the *Simple Object Access Protocol*) for sending around request whereas their interfaces are described by the *Web Service Description Language* (*WSDL*).

As in other areas it is becoming more and more popular to use web services in the field of natural sciences for providing access to data repositories or server based algorithmic calculations [53][54]. The plug-in itself adds functionality to the system, that allows new web services to be accessed from within the system easily. As a reference, an integration of the *WSBDbfetch* web service [55] of the *European Bioinformatics Institute* (*EBI*) was implemented. This service allows to query several databases from different fields of biology.

The user interface is designed in form of a wizard guiding the user through the process of query creation and database selection. In a last step, the retrieved data is stored as a *BioResources* in a virtual folder (= temporary folder, whose content will be lost on closing the application) within the navigator view. These resources can then be accessed, opened and manipulated like any other resource in the system.

Another mechanism to transport chemically enriched information via the internet is *CMLRSS* [56]. *CMLRSS* is the chemically-aware extension of the *RSS* protocol. Depending on the version one is referring to, *RSS* stands for *Rich Site Summary* (*RSS* 0.9x) [57][58], *RDF Site Summary* (*RSS* 1.0) [59] or *Really Simple Syndication* (*RSS* 2.0) [60]. Anyway, *RSS* is a platform independent *XML*-based (see Chapter 6.1) format developed for the interchange of news or other web based content. Information in *RSS* is delivered in an *XML* file called "*RSS* feed" or "*RSS* stream".

By extending "normal" feeds of version 1.0 or 2.0 with *CML* (see Chapter 6.2) encoded chemical information, it is possible to easily distribute chemical molecules and related meta data together with *RSS* encoded news or information. The *CMLRSS* plug-in adds tools for handling *CML* enriched news and blog-feeds to the *Bioclipse* framework. From these feeds the chemical information is automatically extracted and converted into the respective *BioResources* and stored in a virtual folder. Therewith, the content is integrated like any other resource, making it possible to display and manipulate the information by using *Bioclipse* editors and views. The resources can be made persistent by just copying them from their temporary folder to any other location within the workspace. All this creates an easy access to chemical information published by using *RSS* feeds.

The *NMRShiftDB* plug-in enables *Bioclipse* to connect to the *NMRShiftDB* database system [61][62]. *NMRShiftDB* is an open (open source and open data) web database for organic structures and their *NMR* (*Nuclear Magnetic Resonance*) spectra (see Chapter 1.3 for further information *NMR* spectra). At the moment of writing, the plug-in offers the user three different functionalities:

1. Submission of datasets to the *NMRShiftDB* database. Therefore, it is possible to enhance existing *SpecMol* resources with literature references using the *BibTeXML* format (a *XML* language for managing bibliographies) and to submit the data using a wizard.

2. Prediction of *NMR* spectra for existing molecules by using the *NMRShiftDB* web service for spectrum prediction based on the database content.

3. Automatic assignment of peaks to shifts via a *NMRShiftDB* prediction.

As most methods and applications developed within this work were integrated into the *Bioclipse* framework, these parts of the application will be explained in detail in Chapter 5.

5 Software and Methods Developed

Within this work a variety of different software modules and methods were created and many of them integrated into the *Bioclipse* framework. They all focus on the management, visualisation and manipulation of chemical data and therewith form the chemoinformatics backbone of that platform.

As structures are the main data type scientists come into contact with on working in chemistry and related fields, these components are all adding functionality, that describes chemical structures more or less directly.

Figure 12 displays the connections and dependencies of most of the different software modules, that were developed for the *Bioclipse* framework within this thesis.

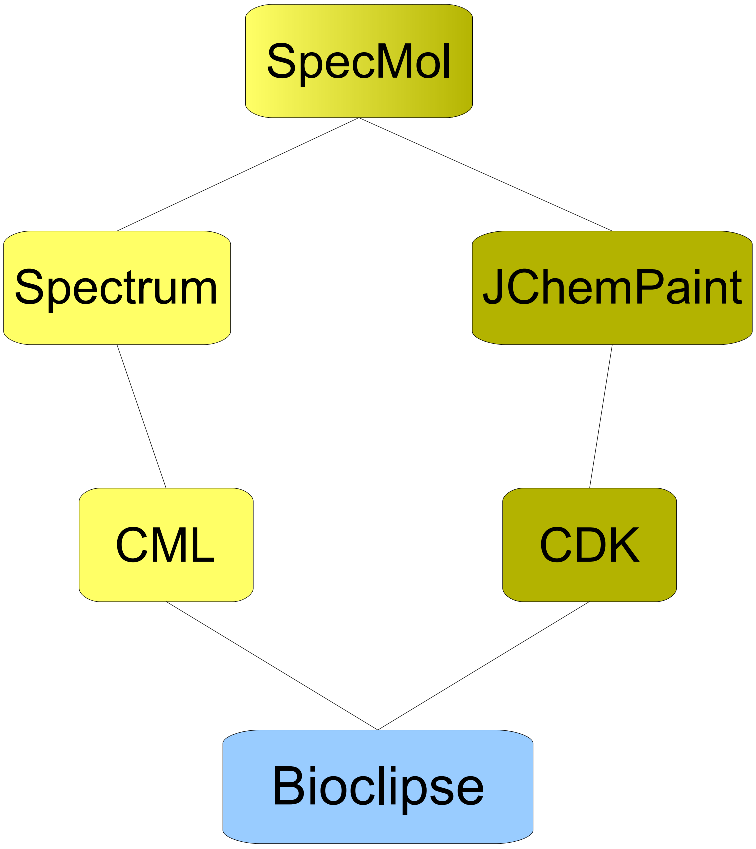


Figure 12: This diagram shows the connections and dependencies of the different chemoinformatics plug-ins of the *Bioclipse* platform. The green marked plug-ins are more focussed on structural information directly, whereas the yellow ones are used for spectrum handling. The *SpecMol* plug-in combines these two fields.

Additionally to the shown components, the necessary framework was built to add a flexible interface to different *Relational Database Management Systems* (*RDBMS*) using object-relational mapping. The developed components can be divided by their function into four distinct categories, that are explained in detail in the following sections. This partitioning does not mean, that they should be seen as separate applications but represents just a logical ordering of the created components. Additionally to this, a data format for encoding spectral information in a highly structured and semantically rich way extending the *Chemical Markup Language* was developed in cooperation with another work group. This vocabulary and the necessary background are described in Chapter 6.

5.1 Structure Handling

The elements described here are primarily focussing on handling structural information. They provide functionality for displaying and editing 2D structural data, are responsible for the correct interconversion of various molecular data formats and integrate components for the visualisation of 3D structures.

5.1.1 The CDK Plug-in

This component integrates the *Chemistry Development Kit* library, a freely available open-source library for structural chemo-, bioinformatics, computational chemistry, and chemometrics, into the *Bioclipse* framework [63]. As the *CDK* library files are included into the generated binary code, any other plug-in within the platform can easily access any method, algorithm or class from the package.

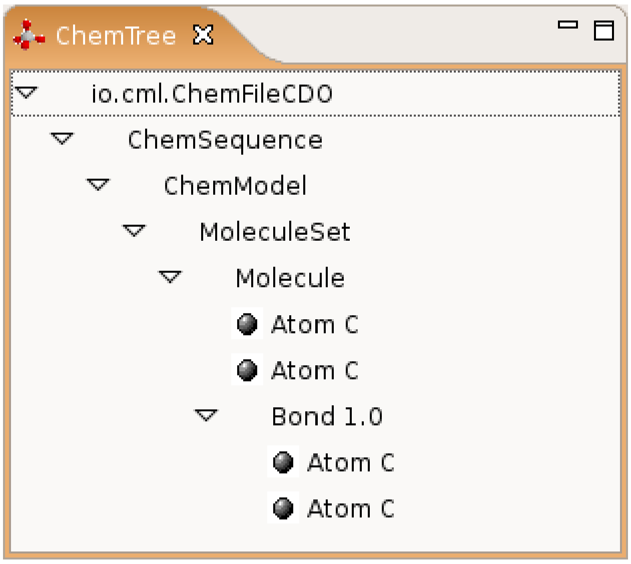
The *CDK* itself provides methods for many common tasks in molecular informatics, including 2D and 3D rendering of chemical structures, I/O routines for different chemical files formats, *SMILES (Simplified Molecular Line Entry Specification)* parsing and generation, *QSAR (Quantitative Structure-Activity Relationship)* descriptor calculation, atom typing, ring searches, isomorphism checking and structure diagram generation.

Many of the other chemoinformatics modules within *Bioclipse* are to a greater or lesser extent using the *CDK* library, as the *CDK* data model for chemical structures is used for the whole platform as internal data structure for the representation of any kind of molecules.

Any objects to be accessed within *Bioclipse*/*Eclipse* are handled as so called resources (see Chapter 3.1.2). The resource for handling structural data within the system is defined in this plug-in - the *CDKResource.* A *CDKResource* is extending the *BioResource* base class by providing methods for parsing molecular data into *CDKMolecule* objects. If a file to be opened contains a collection of molecules, the *CDKResource* separates them and creates sub-*CDKResources* for any of them. The sub-resources are shown within the *BioResource Navigator* as children of the originally activated resource and can be opened separately by double click. If such a sub-resource is changed and saved, the changes are reflected to the parent molecular file and saved there.

Additionally to the methods for parsing molecular data, the *CDKResource* contains several methods to access computable properties of the molecule like its molecular mass, its molecular formula and the *SMILES* string describing this molecule. These properties are displayed for parsed *CDKResources* within an additional information block in the properties view described in further detail later in this section.

As the parsing of molecular structure files is based on the *CDK* I/O functionality, *Bioclipse* is capable to import and export the same molecular formats as the *CDK* itself. These currently include *XYZ*, *MDL* molfile, *PDB* and *CML*.



```
io.cml.ChemFileCDO
  ChemSequence
    ChemModel
      MoleculeSet
        Molecule
          Atom C
          Atom C
          Bond 1.0
            Atom C
            Atom C
```

Figure 13: Screenshot of the *ChemTree* view showing ethane.

Beside the allocation of library files, the plug-in contributes several graphical components to the *Bioclipse* system as well. The *ChemTree* view is adding a tree like structural view on the composition of the actually selected molecules object tree. Figure 13 shows a screenshot of the *ChemTree* displaying the object tree for ethane. This tree gives users and developers a good visualisation of the in memory object representing the actually handled structure file. Additionally, any selection done in this view is reflected to other views used to visualise this structure. So, if the user selects e.g. an atom in this tree, the same atom is highlighted in a possibly open *JChemPaint Editor* and/or an open *2D Structure View* (both explained in detail later in this chapter).

The *2D-Structure View,* as it can be seen in Figure 14, displays 2D depictions of one or multiple structures (a comparative visualisation for three different molecules is shown). If multiple structures are selected to be shown, they are arranged in a table like manner, facilitating the comparability of the shown structures. To allow the display of multiple selected molecules in this view an action was added to the *CDKResources*' context menu. On execution of this action the parsing status of every selected file is been checked and if any of these files are not already parsed this is done. The multi displaying itself works by simply selecting multiple parsed resources, as the *2D-Structure View* is listening to the global selection events and extracts the single molecules from these.

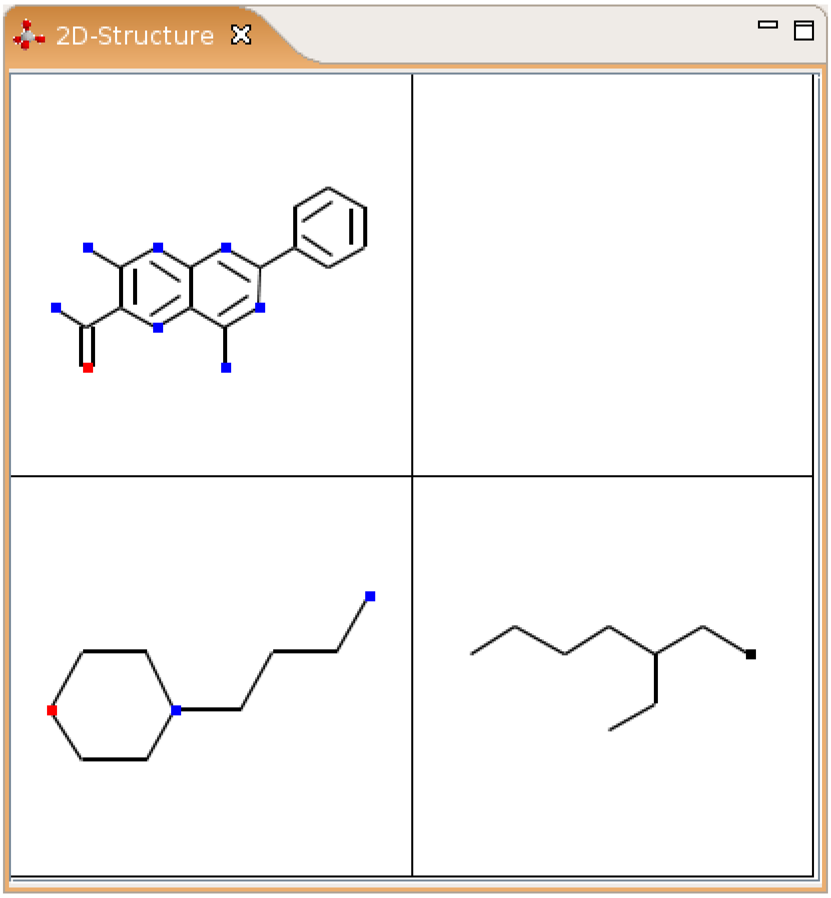
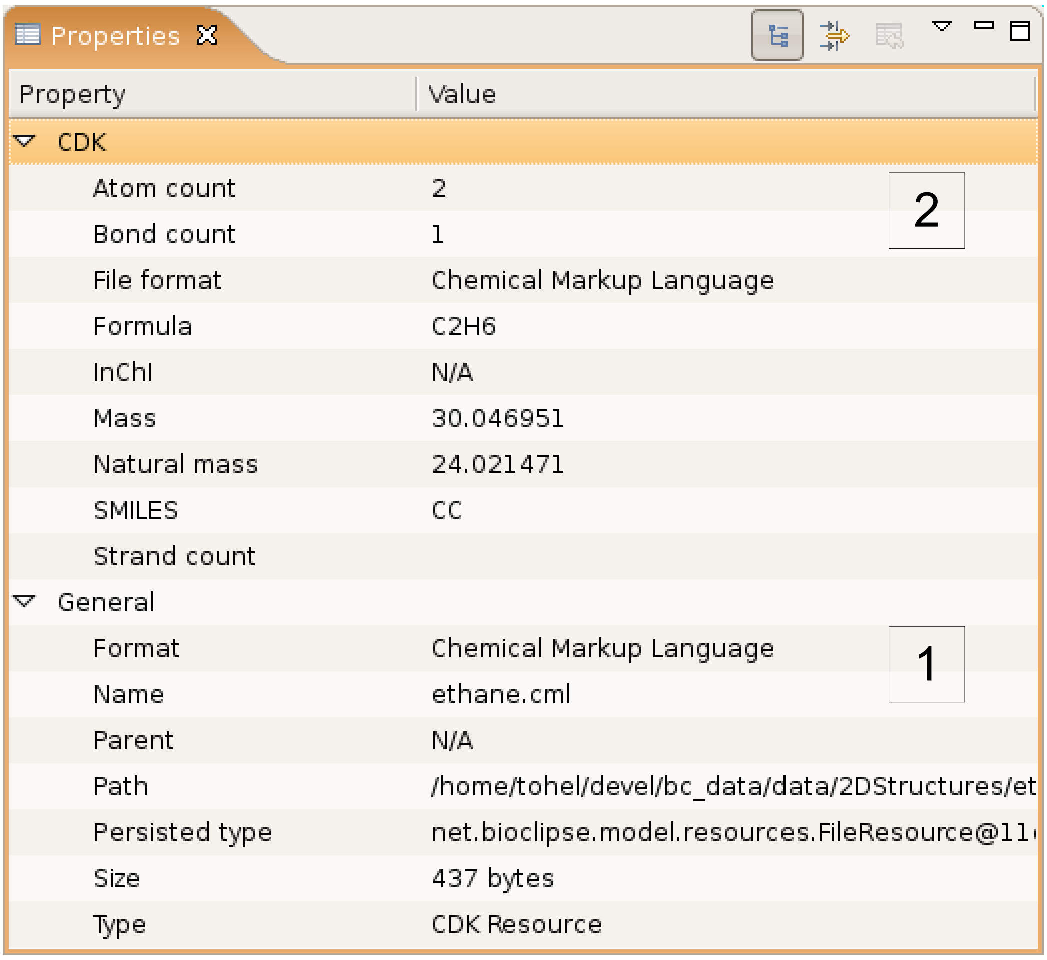


Figure 14: Screenshot of the *2D-Structure* view displaying multiple molecules in a table like manner (4-morpholinepropanamine, 4,7-diamino-2-phenyl-6-pteridinecarboxamide and 1-chloro-2-ethyl-hexane).

On a technical level the displaying of the molecules is realised via the *CDKs* rendering facilities for creating a *Swing* based graphics object of the molecules. In a next step, these graphics objects are embedded into a standard *SWT Composite* using the *Swing* to *SWT* bridging mechanism.

A *Composite* in *Eclipse* is the basic graphical building block for designing *GUIs*. For embedding *Swing* components a special variant of this component is instantiated using the attribute SWT.EMBEDDED. This allows the creation of an *AWT Frame* as child of the *Composite* which can then be filled with any *AWT/Swing* graphical component.



Property	Value
▽ CDK	
Atom count	2
Bond count	1
File format	Chemical Markup Language
Formula	C2H6
InChI	N/A
Mass	30.046951
Natural mass	24.021471
SMILES	CC
Strand count	
▽ General	
Format	Chemical Markup Language
Name	ethane.cml
Parent	N/A
Path	/home/tohel/devel/bc_data/data/2DStructures/et
Persisted type	net.bioclipse.model.resources.FileResource@11
Size	437 bytes
Type	CDK Resource

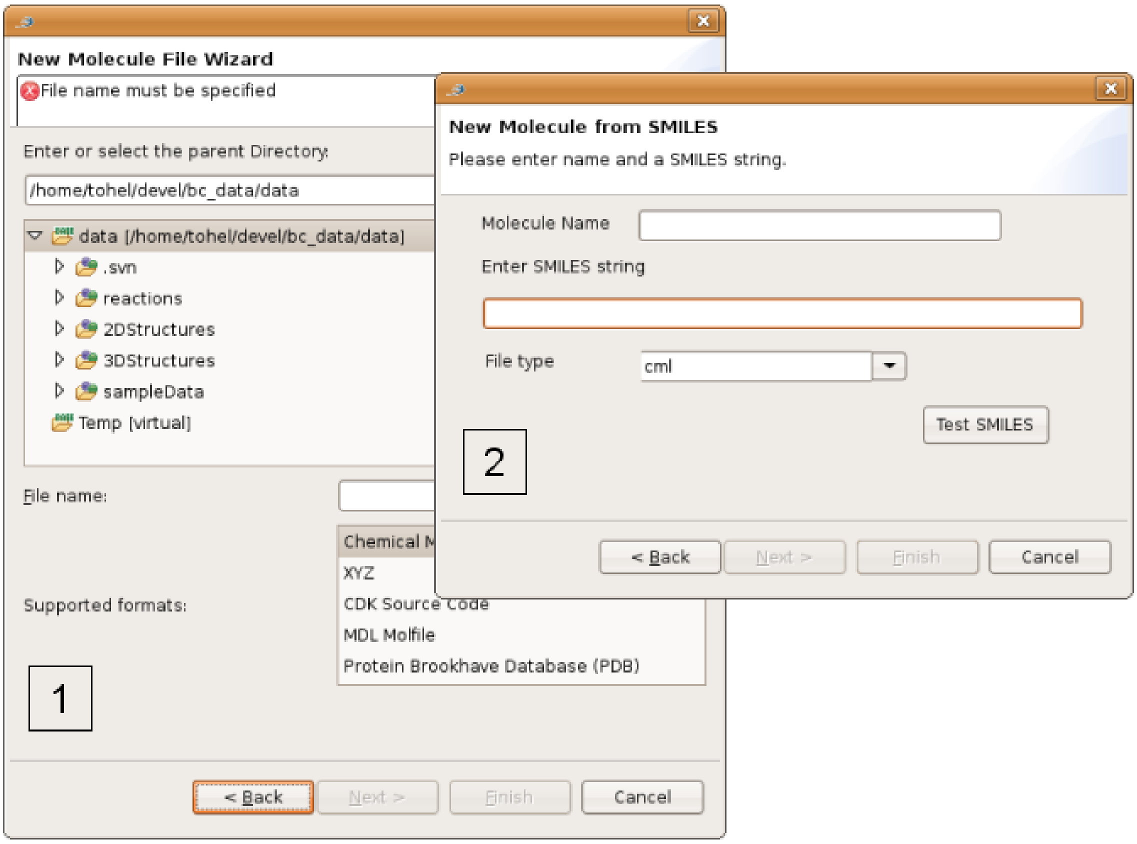
Figure 15: Screenshot of the *Bioclipse* properties view displaying the extended properties for a *CDKResource* additionally to the general ones. Within the section marked with 1, some general information about the selected resource is shown. This is extended by advanced facts of the chemical structure for parsed data (section 2).

As already mentioned, this plug-in extends the *Properties View* provided by *Bioclipse* with an additional block of molecule related information. The general part of this view displays information about the selected resource in general, like its format, its name, the type of resource it is recognised as, and the size of the object (see Figure 15-1). To retrieve this information the resource does not need to be parsed. In contrast to this, the advanced *CDK* properties do need a parsed resource. Just then it is possible to determine information like the atom and bond count, the chemical formula of the molecule, its masses (mass and natural mass) and the *SMILES* string for the molecule (see Figure 15-2).

Beside these graphical elements, directly contributing to the *Bioclipse* workbench, two wizards for the creation of new molecular files are defined. Furthermore, a “Save

As..”-dialog to be used with molecular editors is provided, and some more actions are added to the context menu available for *CDKResources*.

The first wizard (Figure 16-1) creates a new empty file in the selected file format at the defined position within the file system, whereas the second one (Figure 16-2) builds a new molecule from a *SMILES* input string and writes this data in the selected format to the file system.



New Molecule File Wizard

File name must be specified

Enter or select the parent Directory:

/home/tohel/devel/bc_data/data

New Molecule from SMILES

Please enter name and a SMILES string.

File type: cml

Figure 16: Depiction showing screenshots of the two wizards included in the *CDK* plug-in. Picture 1 shows the wizard for new molecule creation, whereas picture 2 shows the wizard for creating a new molecule using a *SMILES* string.

The dialog for saving an existing file in another format looks similar to the wizard for creating new molecules, just that the actual file name is already inserted, but can be changed. By selecting one of the offered file formats the encoding is defined. The list of possible file formats is automatically generated by using a *CDK* mechanism that determines the data formats supported by the current structural data.

As the *CDK* provides algorithms for the calculation of 2D coordinates as well as 3D coordinates, there are two actions integrated into the context menu for *CDK* resources that allow for using these algorithms to extend the existing structural data with 2D/3D coordinates.

5.1.2 Embedding JChemPaint

For enabling *Bioclipse* to display and manipulate 2D structures the *JChemPaint* molecular editor of the *CDK* project is used. *JChemPaint* is open-source, freely available under the *LGPL* license (*GNU Lesser General Public License*), completely written in *Java* and developed by an international team of open-source developers [64].

As this editor is designed using *Suns*' *Swing API*, the drawing components, all menu entries, the toolbar entries and the action handling had to be adapted and integrated into the *SWT* based design of *Bioclipse*. This is realised by using the already described *Swing* to *SWT* bridging mechanism and by wrapping around the existing listener model of the context menu. This ensures smooth and problem-free access to the context menu, which is *Swing* based as well.

JChemPaint is the main editor for chemical structures within *Bioclipse* and used for visualisation of all *CDKResources* containing 2D structures. The editor itself inherits all the features the standalone editor provides. These were extended for the application to seamlessly integrate into the overall framework. Therewith it provides the user with all the functionality expected by a modern 2D editor for molecular structures (e.g. drawing of bonds and atoms, selection of ring templates, import/export from/to various file formats, flipping and rotating of selected parts of a molecule, stereo descriptors, and import/export of *SMILES*).

The editor itself is designed as an *Eclipse MultiPageEditor* showing the underlying data model in two different ways (see Figure 17). The first is the graphical representation of the structure as a graph using *JChemPaint*, whereas the second representation is showing the source of the respective file in the regarding text editor. These two editors are in sync to each other at any time, so that changes in one are immediately reflected to the second one.

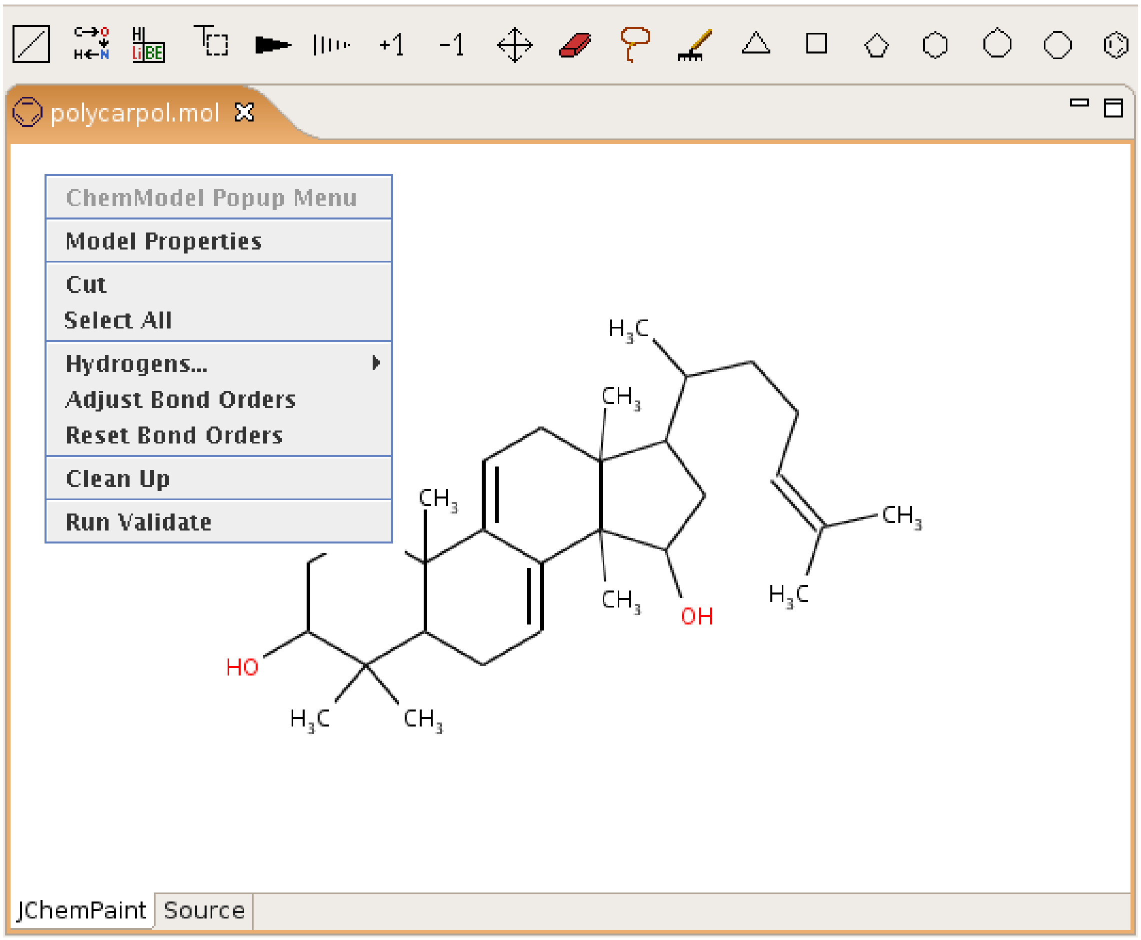


Figure 17: Screenshot of the embedded *JChemPaint* editor showing the 2D structural representation of polycarpol. Additionally the *Swing* based context menu of the editor is shown. The two tabs at the bottom let the user switch between a text based editor showing the original file content and the *JChemPaint* editor activated in this screenshot. Above the editor itself the toolbar defined by this editor can be seen.

The set of actions provided by *JChemPaint* is created by using *Java* properties files. In these properties files, the tree-like structure of the actions is defined as well as into which global menu they belong, which actions are part of the context menu and which are thought to show up in the toolbar. Additionally, these files define, how the needed resources, like icons and the action classes to be executed, are accessed. The menu and the toolbar are generally editor-dependent in *Eclipse* and therewith just show up if the regarding editor is open and selected. If the editor is just not focussed at the moment the regarding menus and toolbar entries are greyed out and not selectable (see Figure 17 upper part for a screenshot of the *JChemPaint* related toolbar).

In contrast to the global menu entries and the actions contributing to the toolbar, the context menu is not converted to a *SWT* menu, but shows up as a *Swing* based interface. This is necessary, because the underlying panel to which the structure is drawn is a *Swing* panel as well and the drawing of *SWT* based menus on top of an embedded *Swing* panel were discovered not to work properly.

All dialogs provided with the *JChemPaint* editor, e.g. for adjusting properties, enter settings for model validation and direct insertion of *SMILES* strings were ported to use *SWT* graphics objects.

The integration of these actions into the system was realised by adapting the base class for all *JChemPaint* actions to extend a *JFace* action instead of a *Swing* based action. In addition to this, it was necessary to adapt the name of the executed method in every action to fit the *JFace* naming schema.

Beside these changes, the embedded *JChemPaint* component was adapted to the undo/redo mechanism of the *Eclipse*/*Bioclipse* framework so that undoable actions are pushed onto the *Bioclipse* undo/redo stack. This was realised by preserving as much of the original undo/redo structure of the standalone *JChemPaint* application, which was implemented in the course of this thesis as well.

Additionally, the editor was adapted to the global copy, cut and paste functionality, so that it is possible to select a (sub-)structure in an open editor, cut or copy it. Theoretically, it should be possible to paste this structure to any other editor that is registered to the clipboard handling facilities within *Bioclipse*. At the moment, this is just implemented for unproblematic exchange between two open instances of the *JChemPaint* editor, but will be implemented at least for text based editors, too. This will be realised by converting the structural information to *InChI (IUPAC International Chemical Identifier)* or *SMILES* and add this at the selected position into the text file.

With this design, it was possible to inherit the general rendering and controlling logic used for the creation of graphical representations of molecular structures from the *JChemPaint* project. As a result of this it will be easy to adapt the embedded component to future changes in the *JChemPaint* project and the adapted *JChemPaint* smoothly integrates into the *Bioclipse* framework and its resource model. It is easy to create molecules in many different formats, display and modify existing structural data and integrate large structural data sets into the framework.

5.2 Spectrum Handling

The modules described here are focussing on the visualisation, manipulation and encoding of spectral information and are used for integrating this functionality into the *Bioclipse* framework. Furthermore, they contain the logic for reading and writing spectral data to different file formats and provide access to the meta data often associated to spectra. By their nature, these components are, together with the ones formerly described for structural data, the basis for further elements allowing for the association of these two types of data to each other.

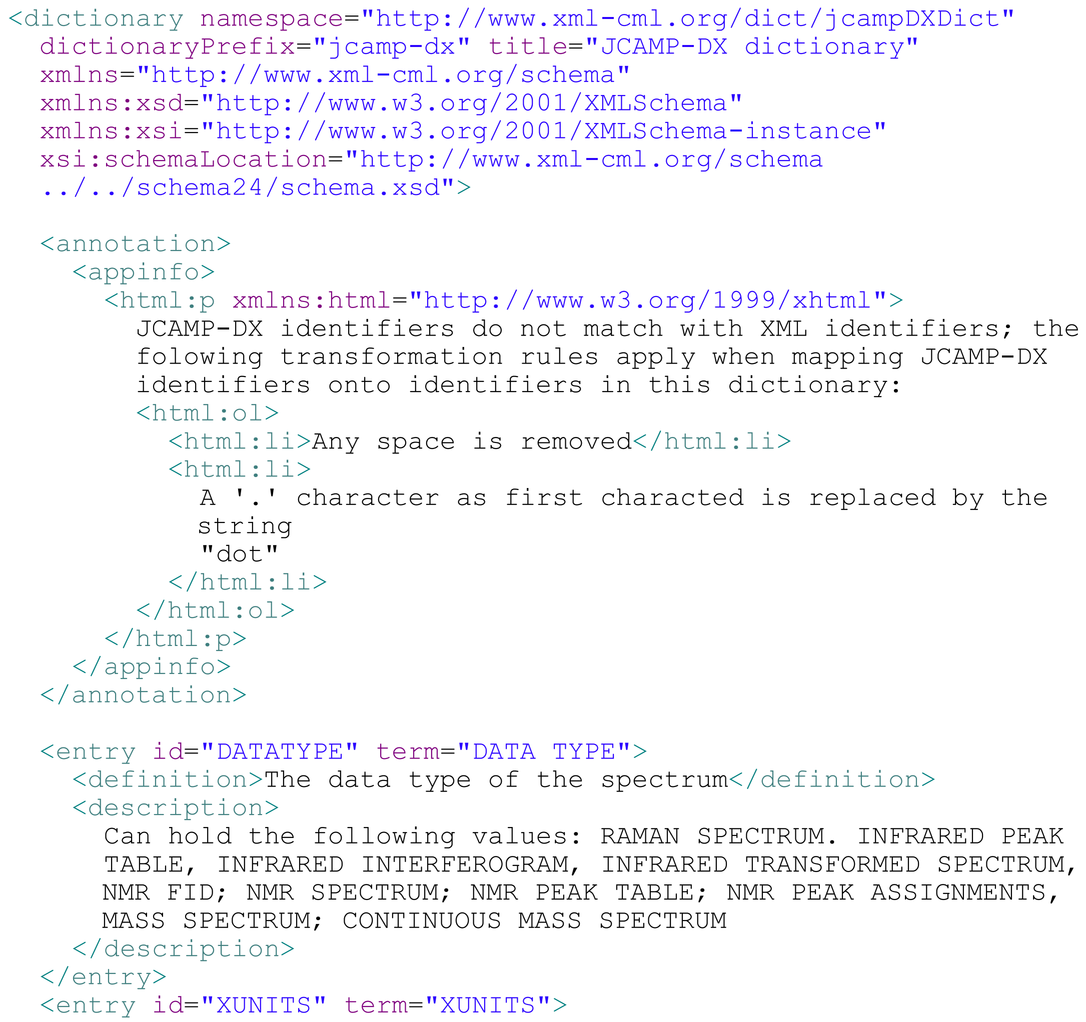
5.2.1 The CML Plug-in

This module provides access to the jumbo *CML* library, an open-source *Java* library for handling and representing *CML Documents* and/or *CML* data structures. *CML* (*Chemical Markup Language*) is a *XML* (*eXtensible Markup Language*) implementation for chemical data/information (also see Chapter 6). *CML* is thought to be an extensible basis for chemically aware markup languages and is structured in a modular way by some core and some extended components. *CML* does share all the general *XML* features and advantages, like being data centric and not presentation centric, being simultaneously human- and machine readable, being platform independent and showing the ability to represent most general data structures [65].

Bioclipse uses the *Java* implementation of *CML* for the internal representation of spectrum data and for the import and export of structures and spectra to and from the *CML* file format.

This component adds a generic resource wrapping around *CML* files to the framework. This resource is extending the *BioResource* base class with methods for parsing *CML* files. It is actually just used for multi-object files, containing more than one parseable *CML* object. In this case the *CML* resource takes care of the extraction of the single objects and creates sub-children for every of these, which can then be selected as any other resource by double-click. If the *CML* resource contains just one spectrum or molecule, the handling of the content is done by the regarding resources for these data types. This is realized by a resource hierarchy, so that any *CML* resource is first tried to be parsed as a spectrum, then as a molecule, and finally as a generic *CML* resource.

Additionally, there is a draft implementation of a *CML* validation, that checks a given *CML* file against the *CML* schema and a variety of dictionaries and outputs any detected errors and warnings to the *Bioclipse* console. The first step, checking the document for validity and well-formedness (see Chapter 6.1 for an explanation of validity and well-formedness of *XML* documents), is done by using the *CMLBuilder* to create a new *CML* document. By doing this the *CML DOM* automatically verifies the well-formedness as well as the validity of the document. In a second step, the whole document is checked for correct dictionary referencing and valid meta data and units names.



```
<dictionary namespace="http://www.xml-cml.org/dict/jcampDXDict"
  dictionaryPrefix="jcamp-dx" title="JCAMP-DX dictionary"
  xmlns="http://www.xml-cml.org/schema"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.xml-cml.org/schema
  ../../schema24/schema.xsd">

  <annotation>
    <appinfo>
      <html:p xmlns:html="http://www.w3.org/1999/xhtml">
        JCAMP-DX identifiers do not match with XML identifiers; the
        folowing transformation rules apply when mapping JCAMP-DX
        identifiers onto identifiers in this dictionary:
        <html:ol>
          <html:li>Any space is removed</html:li>
          <html:li>
            A '.' character as first characted is replaced by the
            string
            "dot"
          </html:li>
        </html:ol>
      </html:p>
    </appinfo>
  </annotation>

  <entry id="DATATYPE" term="DATA TYPE">
    <definition>The data type of the spectrum</definition>
    <description>
      Can hold the following values: RAMAN SPECTRUM. INFRARED PEAK
      TABLE, INFRARED INTERFEROGRAM, INFRARED TRANSFORMED SPECTRUM,
      NMR FID; NMR SPECTRUM; NMR PEAK TABLE; NMR PEAK ASSIGNMENTS,
      MASS SPECTRUM; CONTINUOUS MASS SPECTRUM
    </description>
  </entry>
  <entry id="XUNITS" term="XUNITS">
```

Figure 18: Depiction showing the first part of the *JCAMP-DX* meta data dictionary describing meta data types for *CML* encoded spectral data.

Past validation a dialog is shown stating the result of the procedure and asking the user if the file should be opened in the regarding editor. Additionally, possible error messages generated within the validation process are printed to the *Bioclipse* console. These error messages very often give a good starting point for correcting an erroneous *CML* file. For further versions of the application an automatic highlighting for incorrect segments and automatic correction proposals are planned to be included.

The set of dictionaries can be found in the “dict10” directory within the project. All dictionary files in the subdirectories “simple” and “units” are used for validation. An exemplary extract from the *JCAMP-DX* dictionary is shown in Figure 18.

This dictionary was as well developed during this project and serves as a mapping basis for the correct conversion of *JCAMP-DX* encoded spectral data to *CML* encoded data. Via this dictionary, it is possible to mark spectral meta data within a *CML* file as *JCAMP-DX* compliant or originating. This allows for lossless conversion from one format to the other and back again. In addition, this dictionary is used as information source for the meta data editor, that is part of the spectrum handling module being described in Chapter 5.2.3.

5.2.2 The JCAMP-DX Format

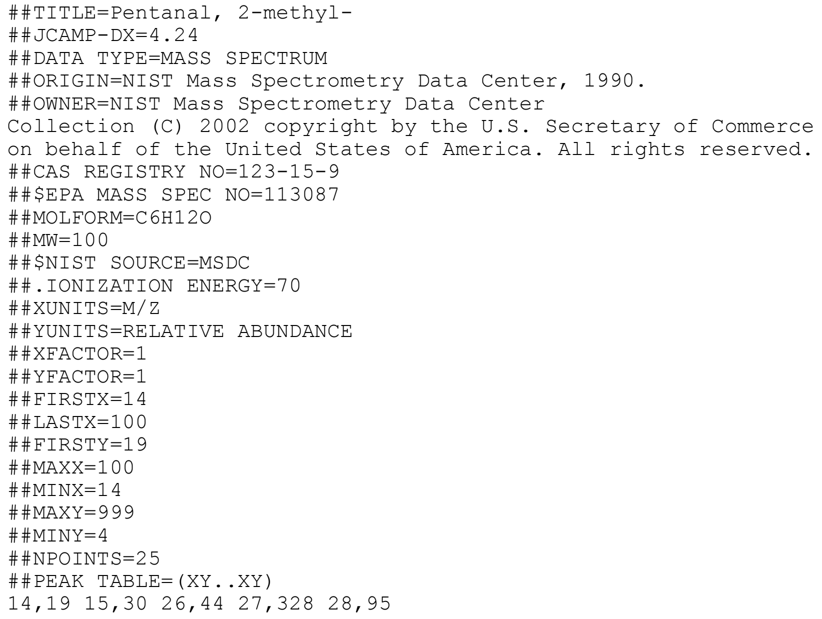
As already mentioned, there is at the moment one other spectral data format supported beside *CMLSpect* (see Chapter 6.3) – the *JCAMP-DX* format.

The *JCAMP-DX* file format is a standardised, portable data format for spectroscopic data, that was originally defined by the *Joint Committee on Atomic and Molecular Physical Data* (*JCAMP*) with the scope to “generate, collect, evaluate, edit, and approve the publication and encourage the distribution of atomic and molecular physical data in suitable form to serve as references for pure compounds and mixtures” (personal communication from Bob McDonald). The *JCAMP* (the organisation) was founded as a task force at the *Pittsburgh Conference* (*Pittcon*) in 1983. Its objective was to design a standardised file format for the exchange of infrared (*IR*) spectra, that should be independent of the used spectrometer and vendor specific software. Beside this, the development was aimed at creating a format that enables long-term archival of spectroscopic data, even past the expected lifetime of current hard- and software [66].

In 1995 the responsibility for the *JCAMP-DX* scientific standards was moved to the *IUPAC* (*International Union of Pure and Applied Chemistry*). Beside *IR* the *JCAMP-DX* format today supports the storage of *UV/Vis* (*Ultraviolet-visible*), *NMR* (*Nuclear Magnetic Resonance*) and *MS* (*Mass Spectroscopy*) spectra as well [67] [68] (for further information on spectroscopy see Chapter 1.3).

As all spectral data is stored as labelled fields of variable length using *ASCII* (*American Standard Code for Information Interchange*) characters, it is human readable and can be edited and annotated using standard text editors (see Figure 19 for an example of a *JCAMP-DX* encoded spectrum). The format itself shows many advantages compared to proprietary vendor formats. It is open-source (see Chapter 1.4.1) and using standard terms, enabling the free exchange of data from any instrument as well as simulated data. There is non-proprietary software available for file conversion from/to other spectroscopic data formats, for internet transmission and for the visualisation of *JCAMP-DX* encoded data.

As the *JCAMP-DX* format is historically grown, there are some drawbacks introduced due to timely evolution and generalisation of the original *IR* focussed definition. For this reason, the definition of the specification became complicated and incomplete.



```
##TITLE=Pentanal, 2-methyl-
##JCAMP-DX=4.24
##DATA TYPE=MASS SPECTRUM
##ORIGIN=NIST Mass Spectrometry Data Center, 1990.
##OWNER=NIST Mass Spectrometry Data Center
Collection (C) 2002 copyright by the U.S. Secretary of Commerce
on behalf of the United States of America. All rights reserved.
##CAS REGISTRY NO=123-15-9
##$EPA MASS SPEC NO=113087
##MOLFORM=C6H12O
##MW=100
##$NIST SOURCE=MSDC
##.IONIZATION ENERGY=70
##XUNITS=M/Z
##YUNITS=RELATIVE ABUNDANCE
##XFACTOR=1
##YFACTOR=1
##FIRSTX=14
##LASTX=100
##FIRSTY=19
##MAXX=100
##MINX=14
##MAXY=999
##MINY=4
##NPOINTS=25
##PEAK TABLE=(XY..XY)
14,19 15,30 26,44 27,328 28,95
```

Figure 19: This depiction displays a section of a *JCAMP-DX* encoded mass spectrum. The file was downloaded from the *NIST Chemistry WebBook* and encodes the peak mass spectrum of 2-methyl-pentanal.

Because of the wide range of data (instrument data, peak table data, chemical structures, ...) integrated successively into the format, software developers ran into the problem of correctly interpreting the usage of tags for writing out their data [69].

Additionally, many vendors of commercial spectroscopic software extended the existing specification by their proprietary fields, leading to a lack of proper documentation of the so used labels. This resulted in a possible incompatibility of *JCAMP-DX* files exported by different machines.

A way to avoid these problems would have been to add applications and routines for proper testing of files stating to be *JCAMP-DX* compliant, so that every provider would have had the chance to directly test his/her implementation.

Nevertheless, essentially all current software for handling spectroscopic data provides routines for import and export of *JCAMP-DX* files and it is the most widely used exchange format for spectroscopic data at the moment.

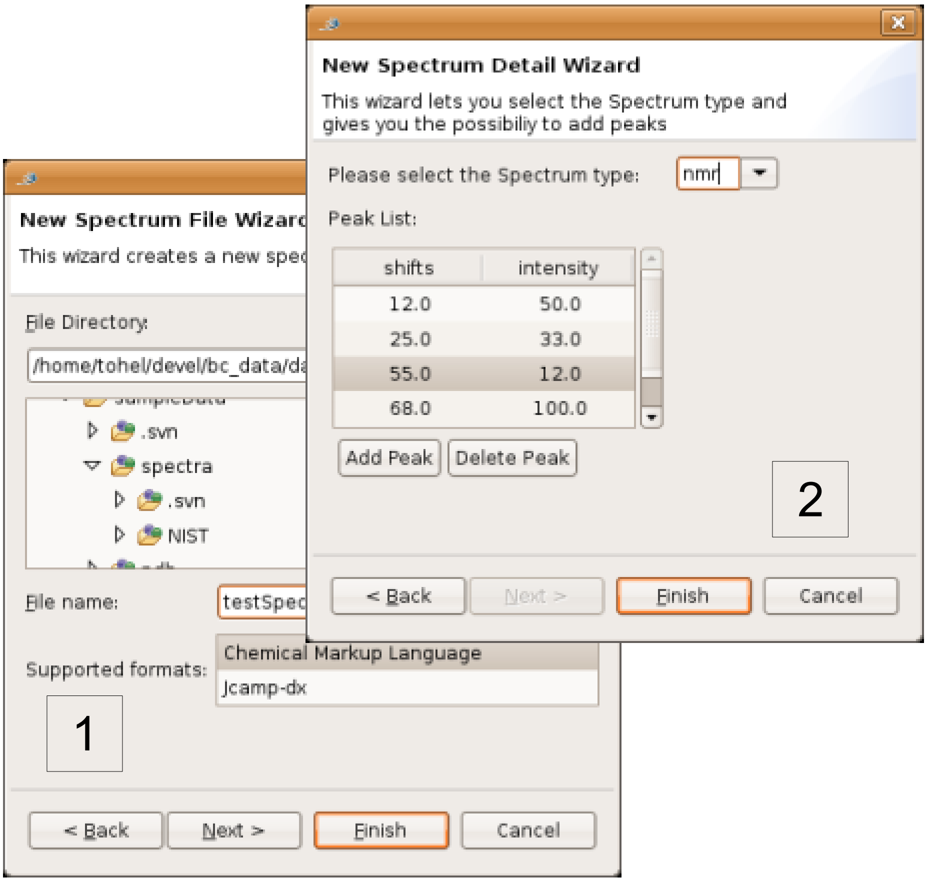
5.2.3 General Spectrum Support

As already stated in Chapter 1.3, the analysis of spectroscopic data can provide scientists with various information about the studied molecules. As one of the major focusses of the *Bioclipse* platform is the support for handling molecular data in diverse manifestations, supporting spectral data is necessary. Therefore, the here described methods and algorithms were developed and integrated into the application. They provide functionality for the integration of spectral data types into the system, provide visual interfaces for this type of data, and provide tools for the manipulation of spectral information.

For the internal representation of spectral data the *SpectrumResource* was defined. This resource extends the *BioResource* base class with methods for reading and writing spectral data from/to the supported data file formats. At the moment two different data formats are supported: the *JCAMP-DX* format (see Chapter 5.2.2) and the *CMLSpect* format (see Chapter 6.3), which is an extension of the *CML* format for spectral data and was elaborated within this project as well. The *Java* implementation of the latter is used as internal data representation for handling and passing around spectral information. For parsing *JCAMP-DX* files the *JCAMP-DX Java* library, the reference implementation of the *IUPAC JCAMP-DX* spectroscopy data standard [66][67][68], is used. This library was originally implemented by *Creon Lab Control*, but the accountability was taken over by our research group in the course of this project.

These libraries are augmented by routines for the interconversion between the two formats, so that a *JCAMP-DX* encoded spectrum can be converted into the *CMLSpect* representation past loading. All spectrum dependent graphical and non-graphical components are using the *Java DOM* of *CMLSpect*.

To enable the creation of new *SpectrumResources* a wizard was added to the *Bioclipse* resource creation context by using the respective extension point. A depiction of the two pages forming this wizard is shown in Figure 20.



New Spectrum Detail Wizard

This wizard lets you select the Spectrum type and gives you the possibiliy to add peaks

Please select the Spectrum type: nmr

Peak List:

shifts	intensity
12.0	50.0
25.0	33.0
55.0	12.0
68.0	100.0

New Spectrum File Wizarc

This wizard creates a new spec

File Directory: /home/tohel/devel/bc_data/da

File name: testSpec

Supported formats: Chemical Markup Language, Jcamp-dx

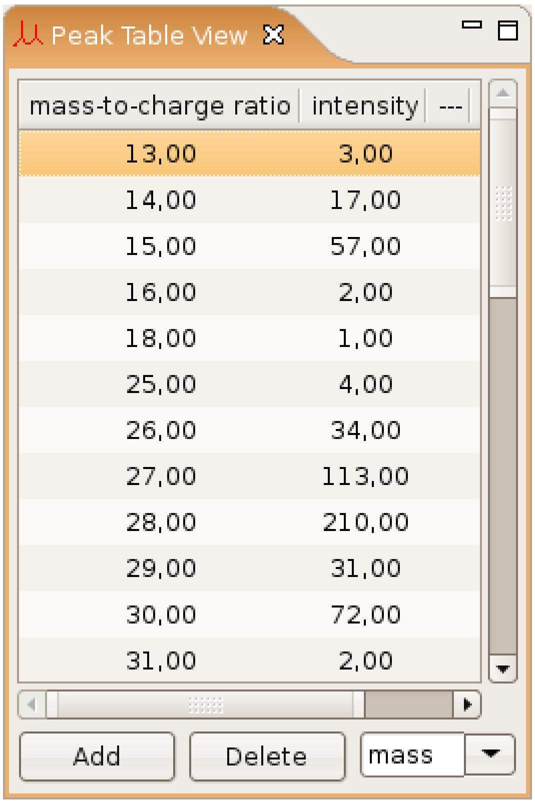
Figure 20: Screenshot showing the two pages forming the wizard used for the creation of new spectra. Page 1 is used to determine the location and format for the new resource, whereas page 2 lets the user define spectrum type and peaks.

Part 1 shows the first page used to locate the place within the currently selected resource tree, where the new resource should be created, to select the spectrum format and to define the name of the resource. The second page allows the user to select the spectrum type and define none or multiple peak signals by entering their x- and y-values (part 2). By finishing the wizard, the data is written in the selected format to the defined file at the chosen position.

For making the source of spectral files available to the user, the text editors provided by the *Bioclipse* core are used. Even though *JCAMP-DX* data is converted to the *CMLSpect* format for internal handling, the user interface shows the original *JCAMP-DX* source in a text editor, whereas *CMLSpect* encoded spectra are shown in the *XML* editor.

Beside this, several views for different representations of the information contained in spectral files were implemented and a perspective was defined, that is per default used for the presentation of *SpectrumResources*. A perspective is a pre-defined arrangement of views and editors (for further information on perspectives see 3.1.3.4). In *Bioclipse* there exist perspectives for handling different types of resources. The regarding perspective is generally provided by the plug-in defining the resource it is bound to. As these perspectives are just proposals how a good arrangement of *GUI* elements might look like, they can easily be overridden by user settings. For anyway giving the user the possibility to recover the pre-defined perspectives, an action was added to the "Windows" menu of *Bioclipse*, which resets any perspective, whose components were newly arranged, to its original setting.

As users might want to have the correct perspective directly loaded on opening a certain type of resource, a general preference model was implemented in the course of this work supporting this. Therefore, a base class handling the integration of the preference pages into the general preferences dialog and providing the graphical backbone was added to the *Bioclipse* core. This class is been extended by implementations for every perspective added to the system. By this preference pages, it is possible for the user to define if a certain resource should any time be represented by using the regarding perspective, if the user should be asked on every occurrence to switch to the regarding perspective, or if the graphical layout of components should stay unchanged, regardless the type of resource being opened.



mass-to-charge ratio	intensity	---
13,00	3,00	
14,00	17,00	
15,00	57,00	
16,00	2,00	
18,00	1,00	
25,00	4,00	
26,00	34,00	
27,00	113,00	
28,00	210,00	
29,00	31,00	
30,00	72,00	
31,00	2,00	

Figure 21: Screenshot of the peak table view of Bioclipse. Visible is a part of the peaks of pyrrolidine, ordered by the mass-to-charge ratio of the peaks.

There are three different views provided, that visualise the actual spectrum information. The peak table view (see Figure 21) shows the peaks of a spectrum in a structured table. This table has two columns for displaying the x-and the y-values and additional columns that can be used for displaying further peak-centric information. The whole set of information can be ordered by any of the columns by just clicking on the columns header.

Additionally, the view displays the type of the spectrum in a drop down box in the lower right corner. This value, as well as the peak values themselves, is editable. For changing the spectrum type the user just has to select another type from the pre-filled drop-down menu. The peak values can be edited by clicking into the regarding table cell and changing its value. Furthermore, it is possible to add a new row by using the “Add” button and any selected row can be deleted by using the “Delete” button. All these changes are directly reflected to the underlying *SpectrumResource* and can be saved to the file using the save functionality of the associated editor.

The probably most common way to represent spectra is by using charts. Two different chart views were implemented for visualising peak spectra as well as continuous spectra - even at the same time, if both data is available. These chart views are using a *Java* charting engine called *JFreeChart* [70]. *JFreeChart* is a completely *Java* based chart library allowing developers to display professional quality charts in their applications. As it is released under a compatible free license (*LGPL*) [29], its use does not add any restrictions to the overall *Bioclipse* license. Figure 22 shows a screenshot displaying a continuous spectrum (*IR* spectrum of dodecyl-benzene) whereas Figure 23 shows a screenshot of the peak spectrum view visualising the *MS* spectrum of pyrrolidine.

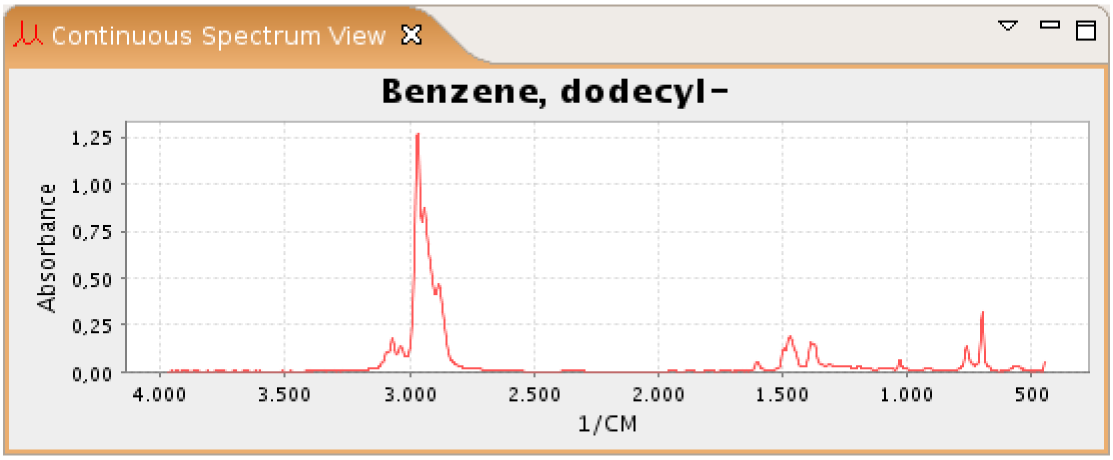


Figure 22: Screenshot of the continuous spectrum view showing an *IR* spectrum of dodecyl-benzene.

The *JFreeChart* based spectrum charts support export to *PNG* (*Portable Network Graphics*) format, a print dialog, a step less zoom of both axes, anti aliasing, and a properties dialog for accessing different chart settings (e.g. change colouring scheme, set fonts, drawing of subsidiary lines).

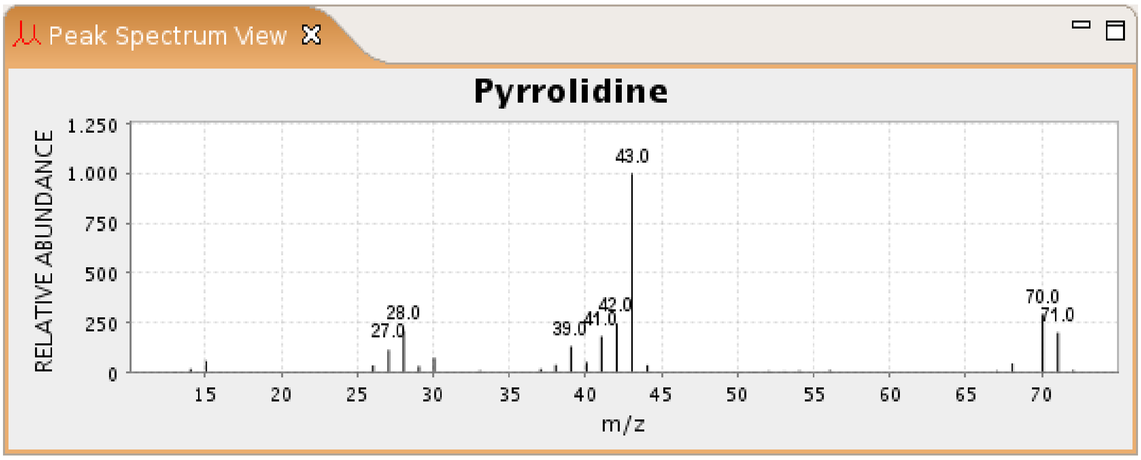
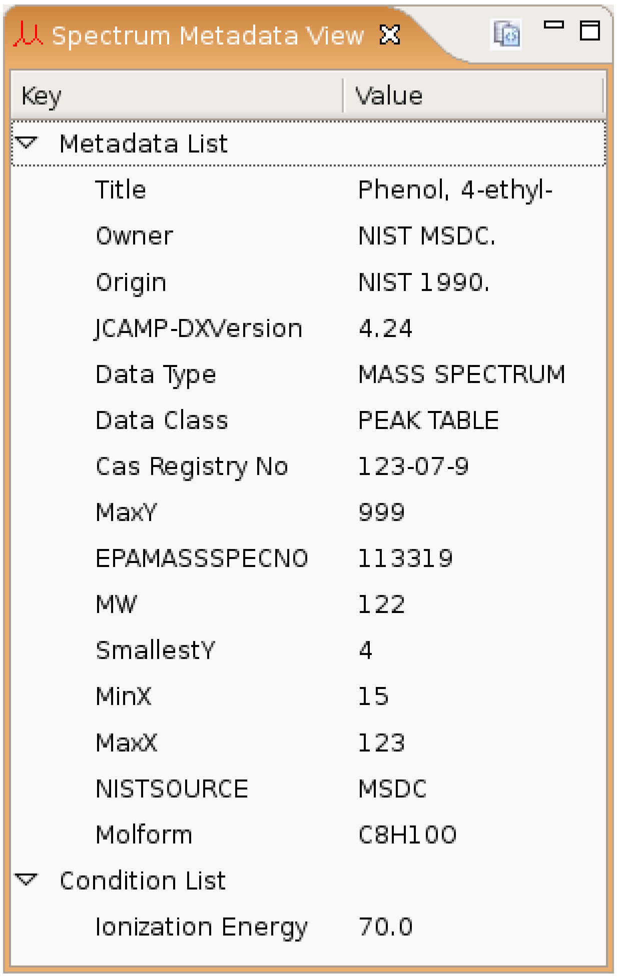


Figure 23: Screenshot of the peak spectrum view displaying the mass spectrum of pyrrolidine.

Furthermore, for peak spectra it is possible to set a threshold that defines which peaks have their x-value printed in numeric form onto the chart. For continuous spectra there exist an extra menu within the view, where a peak picking algorithm can be started. This algorithm works by using a gaussian filter to remove possible noise and a first and second derivative based method for the picking of the actual peaks. The integrated action is an adaptation of an algorithm published in a former release of the *Jumbo CML DOM* [71].



Key	Value
Metadata List	
Title	Phenol, 4-ethyl-
Owner	NIST MSDC.
Origin	NIST 1990.
JCAMP-DXVersion	4.24
Data Type	MASS SPECTRUM
Data Class	PEAK TABLE
Cas Registry No	123-07-9
MaxY	999
EPAMASSSPECNO	113319
MW	122
SmallestY	4
MinX	15
MaxX	123
NISTSOURCE	MSDC
Molform	C8H10O
Condition List	
Ionization Energy	70.0

Figure 24: Screenshot of the meta data view displaying the meta data of a *JCAMP-DX* encoded mass spectrum of 4-ethyl-phenol derived from *NIST*.

Beside this data, that directly defines the spectrum itself, there are often various additional information about the measured substance, the experimental conditions, used substances, etc., recorded and stored as well. This data is referred to as meta data. Meta data does not just help in spectrum analysis and interpretation, but enables scientists to recapitulate the experiment itself and therewith allows for the comparison of results.

To support this, several graphical components were implemented, that allow displaying and manipulation of existing spectrum related meta information. To display the meta data of a parsed spectrum to the user, the *Metadata View* (see Figure 24) is integrated into the spectrum perspective. Within this view all meta information for the selected spectrum is visualised in a structured way using a *SWT TableTree* component. This is a combination of table and tree structure allowing for the arrangement of entries to groups, that show up as collapsible table elements. Generally, the items are assigned to three different groups (meta data list, substance list and condition list), related to the structure of meta data within the *CMLSpect* definition (see Chapter 6.3).

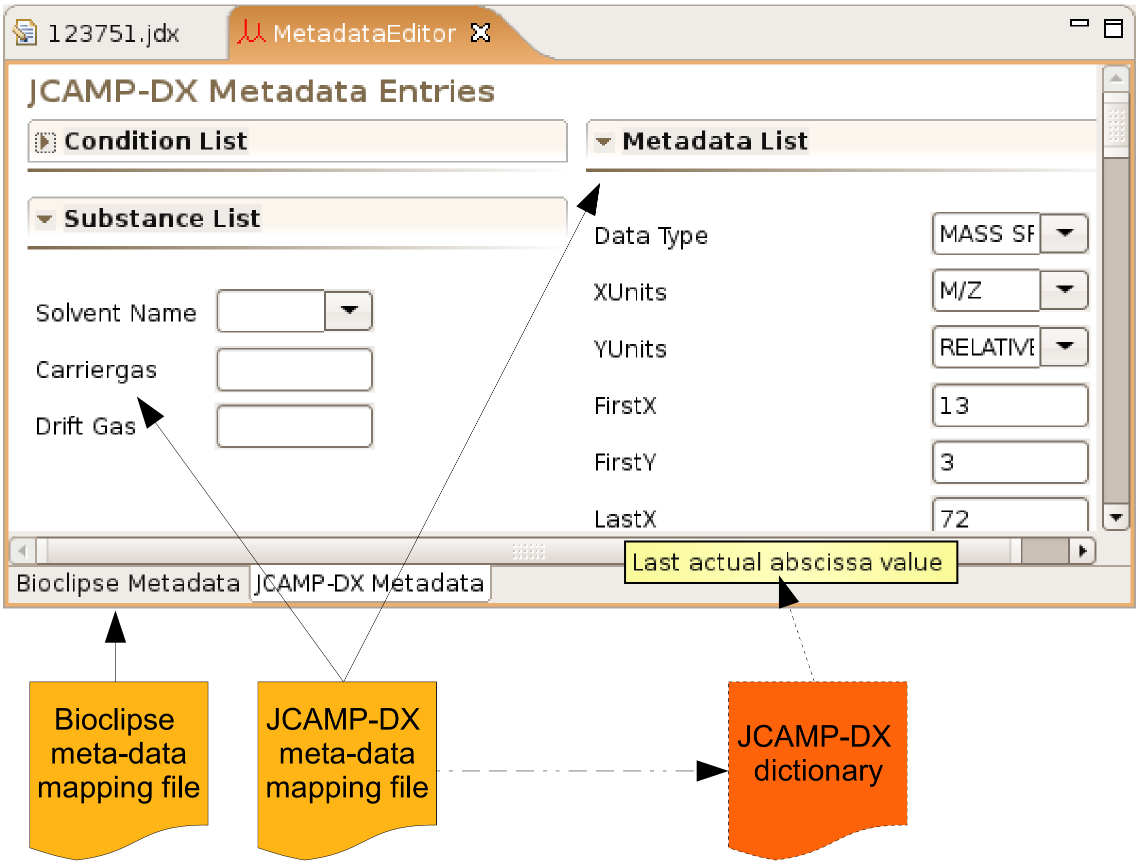
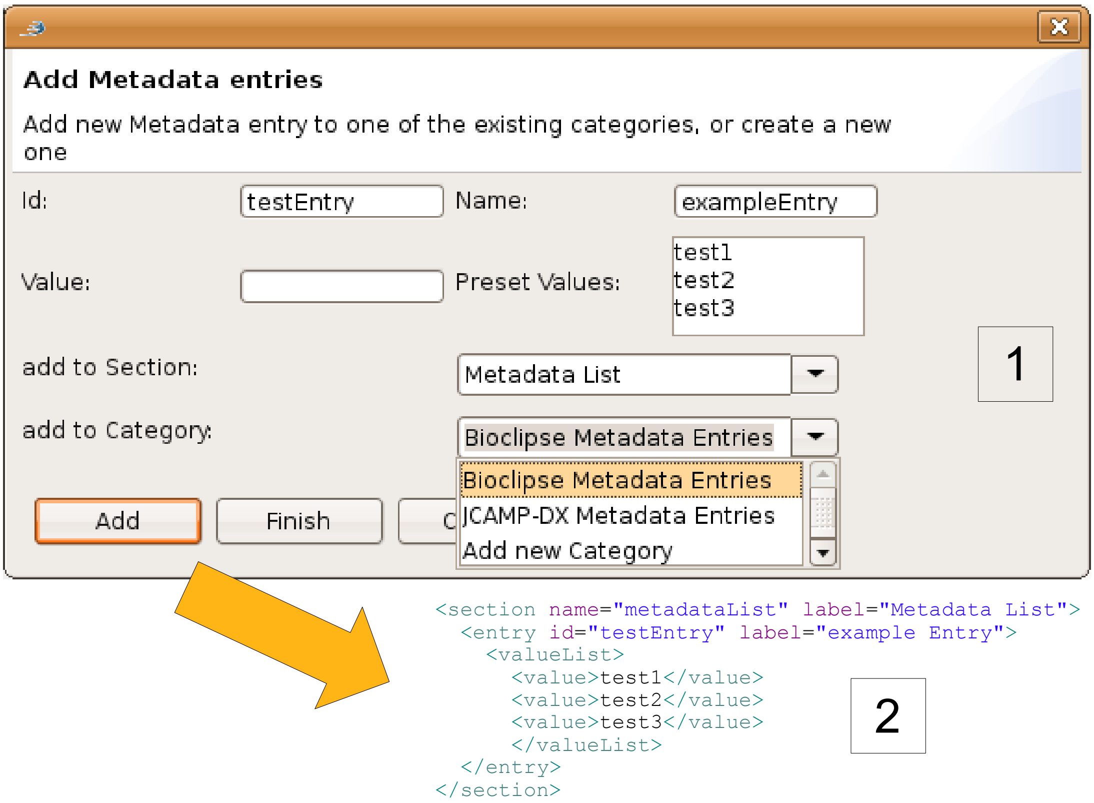


Figure 25: Diagram illustrating the connection between the meta data editor, the meta data mapping files and available dictionary files. Every mapping file defines the categories and entries displayed, whereas an available dictionary is used to generate the tooltips.

Via the regarding action from the views' toolbar, the meta data editor can be opened. This editor is a *MultiPageEditor* using mapping files and available dictionaries to enable the manipulation of meta data entries. The mapping files define the possible entries, whereas connected dictionaries are used for generating descriptions for the respective entries (see Figure 25 and Chapter 9 page 121).

Every page within the editor itself is designed as a structured form by using the *Eclipse Forms API*. The *Forms API* extends *SWT* by exposing a set of custom widgets and other supporting classes, that allow for the creation of polished, 'web-like' *UIs*. The editor is structured using the same scheme, which is used within the *Metadata View*.

The dependency of the data visible within the meta data editor from the defining mapping and dictionary files is shown in Figure 25. For every existing mapping file, there is one page within the *MultiPageEditor* set up, with all entries defined in the related mapping file. Existing values within the spectrum are read out and pre-set in the editor fields. If the mapping file is directing to an available dictionary, this dictionary file is used to generate tool tips for the displayed entries.



Add Metadata entries

Add new Metadata entry to one of the existing categories, or create a new one

Id: testEntry

Name: exampleEntry

Value:

Preset Values: test1 test2 test3

add to Section: Metadata List

add to Category: Bioclipse Metadata Entries

```
<section name="metadataList" label="Metadata List">
  <entry id="testEntry" label="example Entry">
    <valueList>
      <value>test1</value>
      <value>test2</value>
      <value>test3</value>
      </valueList>
  </entry>
</section>
```

Figure 26: Illustration showing the dialog used for adding new meta data entries to an existing mapping file (section 1) and the resulting entry within that file (section 2).

In order to enable users to add new meta data entries to an existing mapping file, there is a dialog available, that can be started via an action, added to the global toolbar by the meta data editor. Figure 26-1 shows a screenshot of this dialog, whereas Figure 26-2 displays the entry created past adding it to an existing mapping file. If multiple pre-set values are defined for the entry, these will be shown in the editor as selectable items within a drop-down box.

For creating a new mapping file instead of adding the meta data entry to an existing one, the item “Add new Category” in the dialog should be selected. This opens another dialog asking the user to enter a name for that file, a label for the meta data category and an identifier. Based on these entries, a new mapping file is created and saved within the regarding directory of the plug-in containing the newly defined meta data entry.

This all makes it easy for users to define their own meta data sets to be added for example to their in house generated spectral data, ensuring uniformity of the associated meta information. Finally, this way of handling meta data allows to visualise and manipulate even datasets containing a large number of associated meta data in a concise manner.

5.3 Assignment of Spectral and Structural Data

In the structure elucidation process, very often multiple spectra are measured for one component. These are used for the extraction of structural features and the generation of a candidate structure based on this information. A very common process within this, is the assignment of peaks or signals from the different spectra to structural elements of the candidate.

The applications described up to now do not separately support this. Therefore, a new component was developed, that integrates implemented functionality and extends it with additional features for the assignment of peaks to substructure elements and *vice versa.* In addition, it adds a resource to the system, that is capable of encoding this information in a proper way.

For this reason, the *SpecMolResource* was implemented. It is based on *CML* and uses a *CML* container for holding one *CML* molecule and none to multiple *CML* spectra. The assignment between the different elements is encoded by using the *CML* referencing system (see Chapter 6.2). To easily recognise the special type of these *CML* files, the new file suffix “.smr” was introduced. This allows to identify the new resource type even without having it parsed. Nevertheless, the content of these files is *CML* and the *CML* parser is used to read the data. The binding to the new suffix has the additional advantage, that these resources can directly be opened in the regarding editor.

The wizard pages that are used to create new assignment resources are displayed in Figure 27. First a name for the resource has to be entered and the storage location for the “.smr” file has to be selected from the current resource tree (see Figure 27-1). In a next step an existing molecular file in any supported format (see Chapter 5.1.1) can be selected, or an empty molecule can be created and added to the file (see Figure 27-2). In any case, this molecule can later be edited using the embedded *JChemPaint* editor. The data is parsed from the selected file and converted to *CML*, before it is added to the new resource. In the last step, shown in Figure 27-3, the user can add spectra from the current working environment to the resource. Like for molecules, if a non-*CML* encoded file is selected, the content is parsed and converted to *CML*. Finally, past finishing the wizard, the new file is created, the data written to it and the resource opened in the assignment editor to let the user perform the actual assignments.

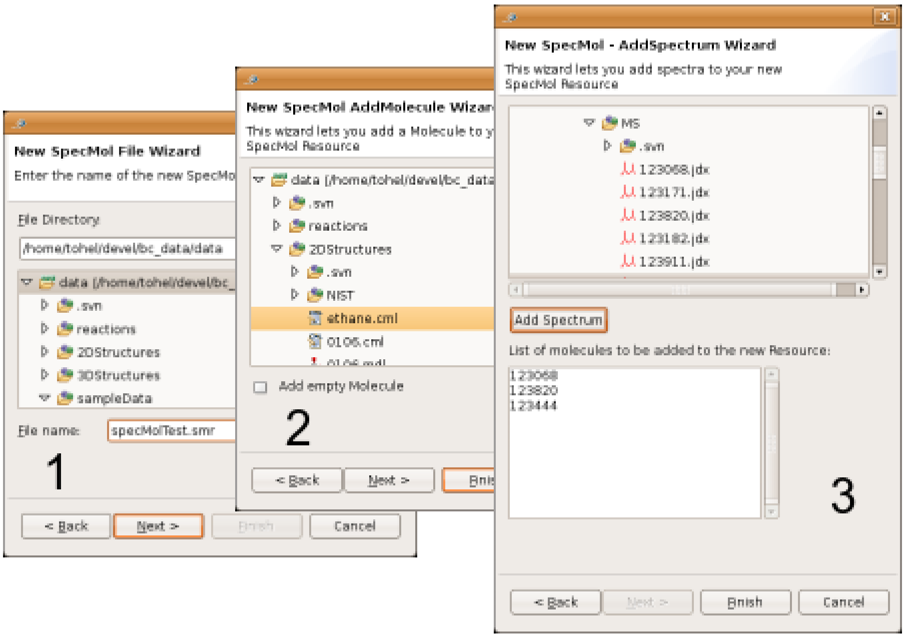


Figure 27: This illustration shows the three pages (1 to 3 reflecting their order of occurence) forming the “new SpecMolResource” wizard. This resources are used for the assignment of structural features and spectral peaks/signals.

Because the *SpecMolResource* is at the moment working with a decoupled child resource schema, there are two additional resources defined. One for contained spectra, that is extending a *SpectrumResource* (see Chapter 5.2.3) and another for the molecule, extending a *CDKResource* (see Chapter 5.1.1). On parsing a “.smr” file, the included spectra and the molecule are extracted and the regarding child resources are created (see Figure 28).

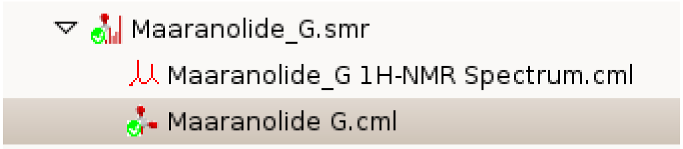


Figure 28: Screenshot of a section of the resource navigator displaying an assignment resource and its child resources, which are generated and added on parsing.

Both these child resources override the “save” methods of the classes they are extending. Therefore, on saving the child, the applied changes are reflected to the parent *SpecMolResouce* and its “save” method is executed resulting in the whole object with all changes being written to the file (see Figure 29).

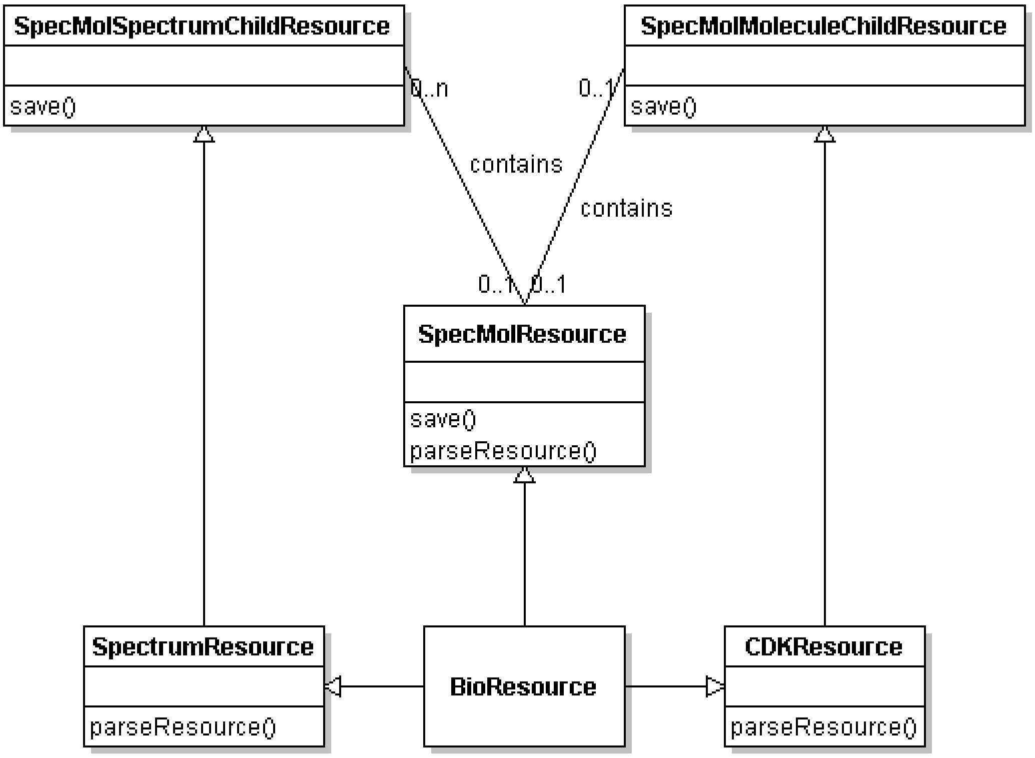


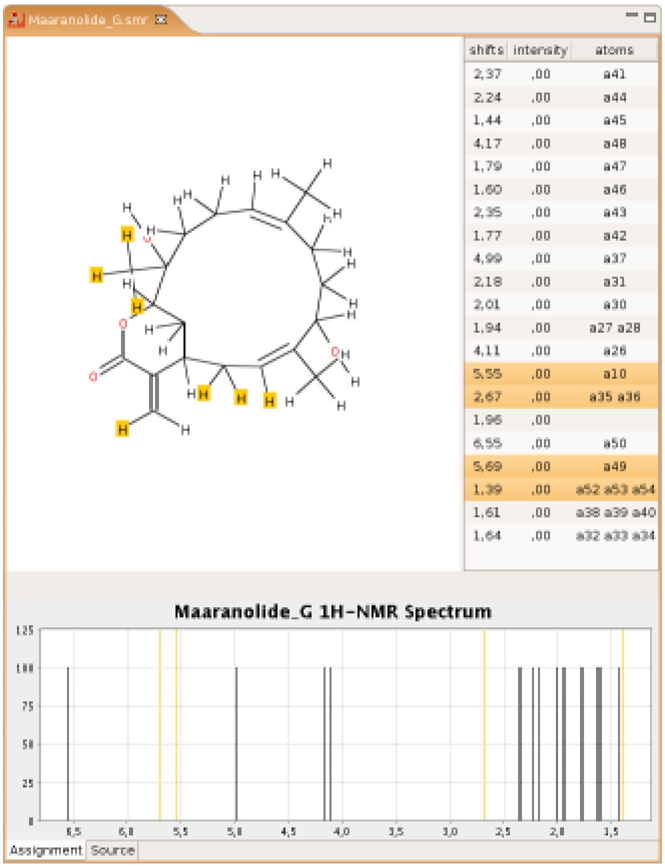
Figure 29: Class diagram illustrating the dependencies of the resources in the assignment plug-in. The *SpecMolResource* extends the *BioResource* base class. A *SpecMolResource* can contain 1-n spectra and 0-1 molecules, that are parsed into the regarding child resources. These in turn are extending *SpectrumResource* and *CDKResource*. As illustrated, the parsing is done by these resources and on saving, the child resources reflect their changes to the *SpecMolResource* parent and the parents save method is executed.

For the assignment process itself and to display already generated assigned resources, an assignment editor was implemented (see Figure 30). This editor integrates three components already used separately in other plug-ins, the *JChemPaint* editor, the peak table view and the peak chart view, but in a more integrated manner. In a second tab, the *CML* source is available in a read-only mode using the *XML* editor.

On loading a *SpecMolResource* the different elements are filled with the regarding content. From the multiple spectra, eventually contained in the resource, just one is displayed at a time. In order to change this selection to another spectrum, the regarding spectrum child has just to be activated by a single click within the resource tree. A double click in contrast, opens the child resource in the correlated editor allowing its manipulation.

On assignment, the editor assigns atoms/bonds to peaks within the *CML* file, by using *CML* references, without doing any interpretation.

The editor has two different modes, that can be changed by pressing the “Switch Assignment on/off” button added to the global toolbar. If the assignment mode is off, the selected spectrum, the structure and the assignments are shown. In contrast to the peak table displayed in Figure 21, the one in this editor shows one more column. In this column the atom numbers of the atoms, which are assigned to the correlated peak, are displayed. If peaks in the peak table or the peak view are selected, they are highlighted in the other one as well. Additionally, the assigned structural elements are highlighted. The same holds for selecting parts of the 2D structure – the assigned peaks are highlighted in the other two components as well. This is realised via a listener model which passes around selection events between the different components.



shifts	intensity	atoms
2,37	,00	a41
2,24	,00	a44
1,44	,00	a45
4,17	,00	a48
1,79	,00	a47
1,60	,00	a46
2,35	,00	a43
1,77	,00	a42
4,99	,00	a37
2,18	,00	a31
2,01	,00	a30
1,94	,00	a27 a28
4,11	,00	a26
5,55	,00	a10
2,67	,00	a35 a36
1,96	,00	
6,55	,00	a50
5,69	,00	a49
1,39	,00	a52 a53 a54
1,61	,00	a38 a39 a40
1,64	,00	a32 a33 a34

Figure 30: Screenshot of the assignment editor showing an assignment exported from *NMRShiftDB*. This example shows the substance maaranolide G and a corresponding ^{1}*H-NMR* spectrum. A subset of the assigned peaks and hydrogen atoms is selected and highlighted in all sub-elements of the editor.

To perform an assignment, the editor has to be switched to the assignment mode. Then it is possible to select one or multiple peaks in the peak table or the peak chart and likewise do a selection of atoms (and) bonds from the structure. By pressing the “Do Assignment” button from the global toolbar the current assignment is stored. After all assignments are done the data can be persisted by just returning to display mode, reviewing the changes, and saving them to the underlying file.

In order to allow for the extension of existing *SpecMolResources* with additional spectra a regarding action was added to the context menu of this resource type within the resource navigator. This was realised by using the corresponding extension point of *Eclipse*.

5.4 Database Connection

The collective of applications, methods and algorithms described so far allow *Bioclipse* to persist data in a variety of different file formats within the file system. Additionally, it is possible to connect from within *Bioclipse* via web services to certain online data bases, which provide an interface for the supported web service standards.

However, a general persistence of data elements to a local or remote database system is a common and very often enquired feature of modern software projects. Persistence to a database system enables for faster searching in large data sets and generally increases the data access speed, by using the indexing methods integrated into these systems. Beside this, the management of large file system based data collections needs more maintenance and strategic planning by the users themselves, whereas by using a database system this is mostly pre-defined by the developers. Furthermore, due to the accessibility of data stored within the file system by any program, the danger of data loss is relatively high. In contrast to this, databases are often included into a centralised backup strategy and are most often just accessible via the connecting software itself, which makes it easier to control user actions.

As *Bioclipse* is a framework with a very broad range of applications, the implementation of the database connection should support this generality as well. The resource scheme implemented in *Bioclipse* is based on persisted resources, that define the connection to either the file or a in memory string and the actual resource objects, which contain the parsed (meaning the interpreted) object of the persisted resource.

Including a connection to a database system would mean to translate the data object to a new persistence object to be stored in that database system. As there are many different database systems used for storing data with relevance for *Bioclipse* and to keep the platform as flexible as possible we decided not to focus on one single *RDBMS* (*Relational Database Management System*), but to introduce a flexible interface enabling the connection to a variety of different database systems.

Modern software projects normally are built on two very different ideas for the object representation inside applications and their persistence in databases. While software in nowadays is mostly implemented using a object oriented programming language, the data storage is done using a relational scheme. The difficulties encountered by transferring information from one to the other is known as the object-relational impedance mismatch.

To bypass these problems, but still remaining flexible regarding the choice of the database management system to use, the mapping of data to the database is realised by using the *Hibernate* object-relational mapping tool. *Hibernate* releases developers of most of the mapping difficulties, by introducing a general mapping scheme based on *XML* mapping files. Additionally, *Hibernate* supports the persistence of data to many different relational database management systems (e.g. *MySQL*, *Oracle*, *PostgreSQL*, *HSQL*) without the need to change the mapping or the *Java* code on changing from one system to the other. To connect to a certain *RDBMS* and persist data to it, *Hibernate* just needs access to the regarding database driver, some connection information and a *XML*-based mapping definition.

Within the next sections, the general concepts of relational database systems and object-relational mapping will be explained, followed by a detailed description of the way the database connection was realised for the *Bioclipse* framework.

5.4.1 Database Systems & Object-Relational Mapping

In today's software development projects normally two very different techniques are used. On the programming level, the object oriented approach is favoured by using languages like *Java*, *C#* or *C++*. For data management and data persistence however, *Relational Database Management Systems* (*RDBMS*) are the technique of choice. The set of conceptual and technical difficulties encountered on trying to translate these two schemas to each other is known as the object-relational impedance mismatch.

5.4.1.1 Relational Databases

In relational databases, data is stored in a series of dependent tables. Every row in these tables is representing one data entry, has a fixed data type and is identified by a unique primary key. This primary key might be generated by just one or by a combination of multiple columns. The data types usable within a database are defined by the *RDBMS* and cannot be altered. The connection of the relational data to each other over multiple tables is realised via so called foreign keys. These are keys used in one table as primary key and in an other table are entered, additionally to the primary key, to define a relationship between these two entries [72].

The primary interface to a *RDBMS* is realised via the query language *SQL (Structured Query Language),* which is a generalised language to formulate database queries implemented by all major *RDBMS*. This language enables the user to do nearly all manipulation necessary by using a small set of commands (create, alter, drop, insert, update, delete and select) [73].

Most *Relational Database Management Systems* are built to be used by multiple users at the same time. This makes it necessary to ensure, that the system is returning meaningful and correct information to any of these users. This is done by using transactions. A transaction in this context can be seen as a container for one or multiple database operations, that do fulfil the *ACID (Atomicity, Consistency, Isolation, Duration)* constraints [73]:

- *Atomicity:* either all tasks of a transaction are performed or none of them
- *Consistency:* at any point the database has to be consistent related to a set of integrity constraints defined by the *RDBMS*
- *Isolation:* operations performed by different users are isolated from each other
- *Durability:* once the success of a transaction has been reported the transaction will persist regardless of e.g. a system failure

5.4.1.2 Object-Relational Mapping

The general idea in object relational (*O/R)* mapping is to introduce an additional layer, that is used for translation and transformation between the two concepts. This can either be realised by individual implementations for the current application or by using one of the existing *O/R* mapping frameworks. These frameworks try to release the developer as much as possible from the mapping process itself and ensure, that the state of an object together with all its attributes is translated to the rows of an relational database table and *vice versa [73]*. A general scheme of this approach is shown in Figure 31.

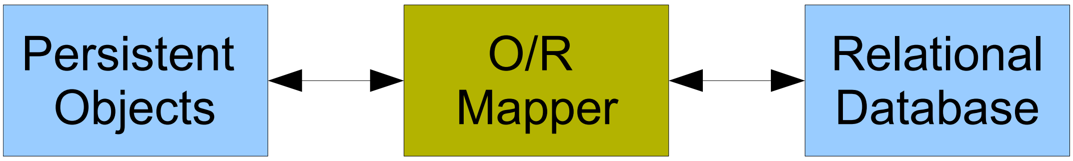


Figure 31: General concept of object-relational mapping.

All this is necessary, because *RDBMSs* store data only, while programming objects have identity, state and behaviour in addition to the data. However, even just storing the data can be a major issue, as there often is no direct mapping between the data types of the used programming language and those of the *RDBMSs*. Furthermore, there is nothing similar to object inheritance and polymorphism in current *RDBMS* and software objects are traversed using direct references whereas database tables are connected via foreign and primary keys [74].

The easiest and straight forward way of mapping objects into a relational database is to perform a one-to-one mapping of the persistent class to a relational table. In this case a column in this table represents one attribute of the class and each class instance is stored in a new row. This schema is in practice sometimes loosened for performance improvement.

The probably most widely used *O/R* mapping system for *Java* is *Hibernate* - an open-source (see Chapter 1.4.1) licensed (*LGPL*) application that stands out for its productive efficiency and flexibility, so that it is applicable to nearly all types of projects [73]. *Hibernate* is described as “a powerful, ultra-high performance object/relational persistence and query service for Java. It lets you develop persistent objects following common *Java* idiom - including association, inheritance, polymorphism, composition and the *Java* collections framework” [75].

Furthermore, it comes with a powerful query language (*HQL* = *Hibernate Query Language*) that is expressed in a *SQL*-like syntax and includes full support for polymorphic queries. However, native *SQL* queries are completely supported as well.

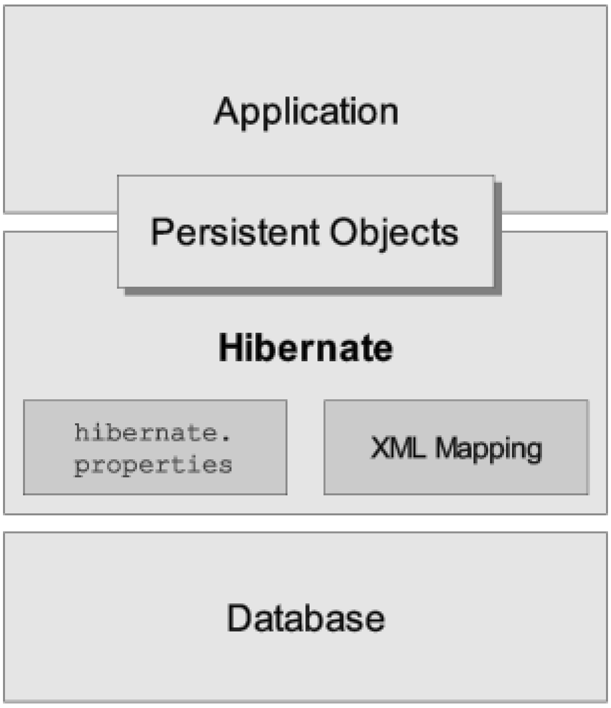


Figure 32: A high level schema of the Hibernate architecture. The diagram shows how the objects of an application are persisted by hibernate to any supported database using mapping and configuration data. (image taken from the Hibernate documentation)

The general architecture of *Hibernate* and its connection to both, the application and the database is shown in Figure 32.

Hibernate uses *XML*-based (see Chapter 6.1) configuration files to connect the relational database to so called *JavaBeans*. *JavaBeans* are *Java* classes, that show a standard constructor, are serialisable and have public get and set methods. For every class, that should be persisted within the database one of these *XML*-based mapping files must exist, describing how this object should be mapped to the database. In these files it is for example defined, which attribute connects to which column and which connections to other tables exist [76].

Additionally, there is one *Hibernate* configuration file, that holds all information needed to realise the actual connection to the selected *RDBMS*, like e.g. which *JDBC* (*Java Database Connectivity* - an industry standard for database-independent connectivity between the *Java* programming language and a wide range of databases) driver to use, which *RDBMS* type is used and other connection relevant information (user name, password, etc.) [76][73].

The usage of *Hibernate* as persistence framework leads to better readable and normally a lot shorter code, improved development time and it makes the application independent of the utilised *RDBMS* [73]. The sole disadvantage is the slightly worse performance, though there are a lot of ways explained on the *Hibernate* web page and in different online forums to improve this as well [75].

5.4.2 Implementation of Database Connections

The integration of the possibility to connect *Bioclipse* to database systems without introducing new dependencies was realised by using the *Eclipse* buddy concept for making the application aware of the database driver. Additionally, three new extension points (see 3.1.1 for further information on extension points and extensions) were defined to pass the configuration information and the mapping data to the central database plug-in. A schematic depiction of this can be found in Figure 33.

This depiction as well clarifies the plug-in schema used for maximal independence of the components from each other. For every resource one database-wrapper module was created as well as for every *RDBMS* to be available. All relevant information is centrally collected within a central database component.

The buddy concept extends the normal class loading policy of *Eclipse*. Under normal circumstances, a plug-in just has access to its internal classes and those imported from dependent plug-ins. However, sometimes it is just not possible to introduce the necessary dependencies. This is what the buddy class loading is used for. By registering a component as a buddy to another one, it exposes itself to it and so enables the access to needed classes even though they are not specifically imported. In *Bioclipse* this mechanism is used to make the central database plug-in aware of the actual *RDBMS* (currently implemented for the *Hypersonic SQL Database*) classes.

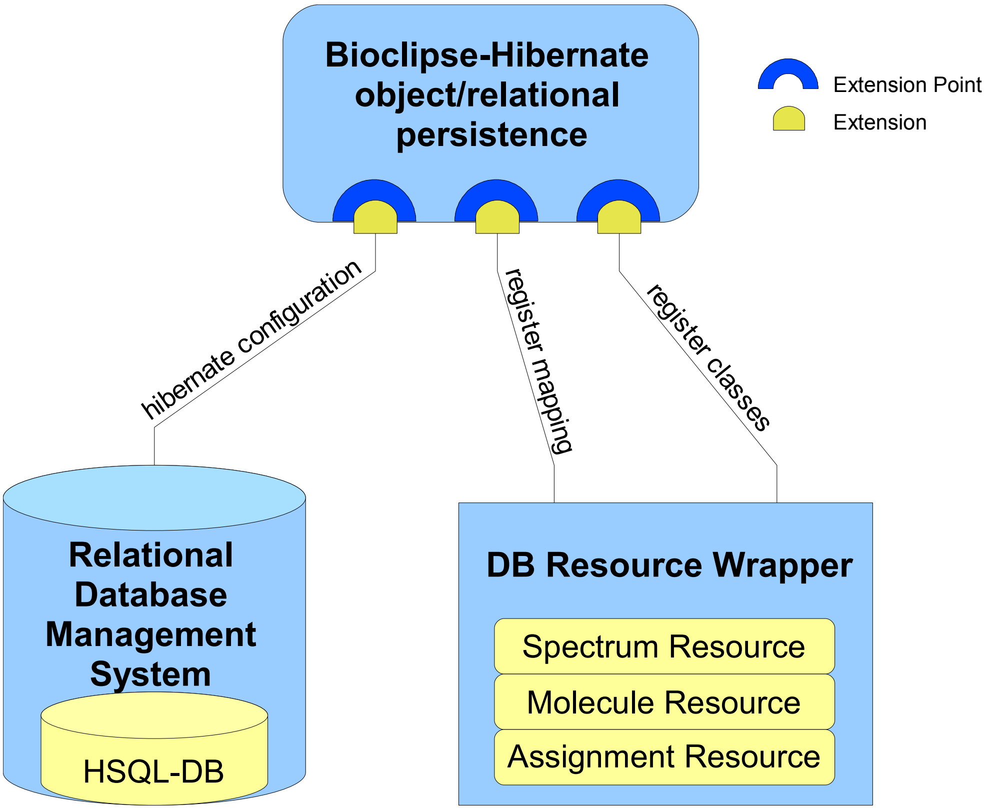
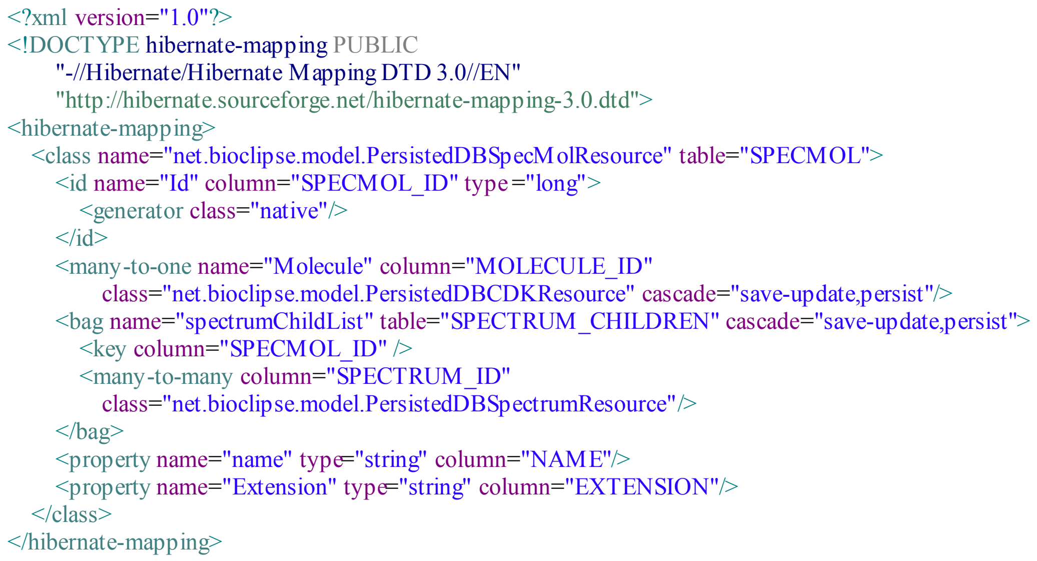


Figure 33: Simplified schema displaying the connection of database systems to Bioclipse and the registration of wrapper classes for the different resources via extension points provided by the central database plug-in.

For transferring the information needed to connect to the database system, the new "HibernateConfiguration" extension point is used. It is, as all newly introduced extension points, defined within the central database component and used to transfer information like the *JDBC* driver class, the *SQL* dialect class, the used sub-protocol, a name used for representing this database system within the resource navigator and a cache provider to hibernate. Except for the name, all this data is needed by *Hibernate* to establish the basic database connection. The other two extension points are directly related to the resources to be persisted in the database. One is used to pass the location of the *XML* based mapping file, defining which resource variables are to be persisted in which manner, whereas via the last extension point the database module is informed which database enabled resource should be used for a given *BioResource* to persist it within the database.

A *Hibernate* mapping file, defining how assignment resources used to represent a molecule, multiple spectra and their assignments should be persisted into relational databases is shown in Figure 34. First of all, a table with the identifier “SPECMOL” is defined for the resource itself. Next, the primary key of this table is set by defining an identifier and map it to the column “SPECMOL_ID” in that table. *Hibernate* requires fields, that should be persisted, to follow the *JavaBeans* syntax (see Chapter 5.4.1.2). As the resource mapped here, contains references to other persistable objects, the next lines define how these should be transferred to the database. For the single molecule allowed within an assignment resource a mapping to a many-to-one relationship in relational databases is defined. The regarding molecule will be saved in the “MOLECULES” table, whereas the identifier is stored as a foreign key together with the assignment resource. The connection of spectra to their hosting assignment resource is representing a many-to-many relationship. *Hibernate* maps this by introducing an intermediate table that contains, beside the id of the assignment object, the id under which the spectrum is stored within the “SPECTRA” table - both as foreign keys.



```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="net.bioclipse.model.PersistedDBSpecMolResource" table="SPECMOL">
      <id name="Id" column="SPECMOL_ID" type="long">
         <generator class="native"/>
      </id>
      <many-to-one name="Molecule" column="MOLECULE_ID"
            class="net.bioclipse.model.PersistedDBCDKResource" cascade="save-update,persist"/>
      <bag name="spectrumChildList" table="SPECTRUM_CHILDREN" cascade="save-update,persist">
         <key column="SPECMOL_ID" />
         <many-to-many column="SPECTRUM_ID"
            class="net.bioclipse.model.PersistedDBSpectrumResource"/>
      </bag>
      <property name="name" type="string" column="NAME"/>
      <property name="Extension" type="string" column="EXTENSION"/>
   </class>
</hibernate-mapping>
```

Figure 34: A *Hibernate* mapping file defining how an assignment resource is to be persisted into a relational database.

Beside this, there are two more string fields stored in the assignment table - the name of the object and the extension for this type of resource. The latter is needed in *Bioclipse* for correct recognition and handling of the resource within the system. Because the assignment resource is actually formed by data provided by these two resources, the object itself is not stored in the database, but reassembled on parsing by the sub elements. An entity-relationship diagram of the resulting database schema for this resource and the other chemoinformatics resources is show in Figure 35.

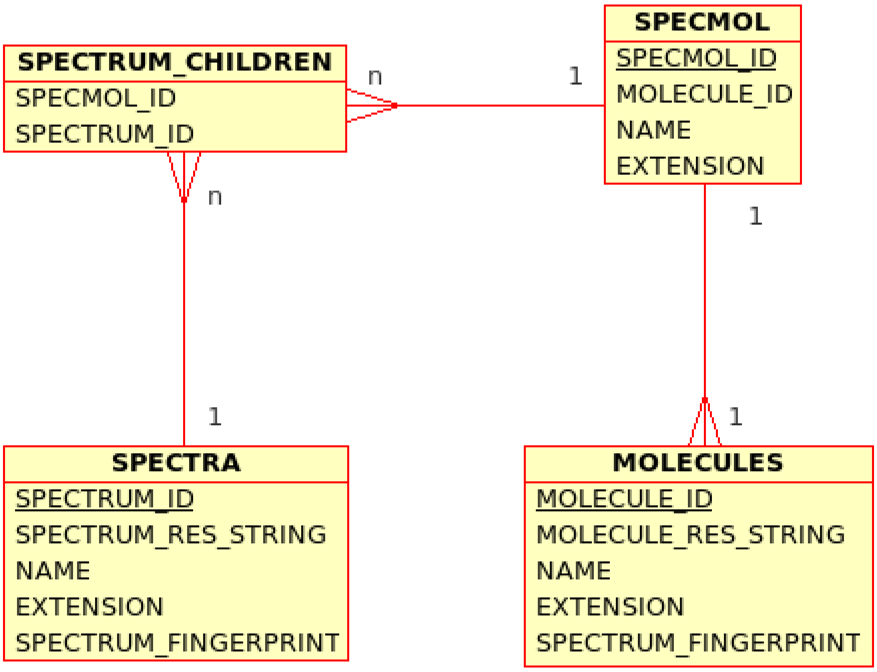


Figure 35: Entity-relationship diagram of the database tables created by *Hibernate* for mapping the three chemoinformatics resources to relational databases.

It was necessary to introduce new resources wrapping around both the *BioResources* and the persistence objects, because beside the string representation of the persisted objects content, additional calculable properties should be persistable to the database. First, these properties are not defined in the existing *BioResources* and second, there exists no connection from the persisted resource to the *BioResource,* which however is needed to pass this additional information from the database via the persisted database resource to the *BioResource* implementation. To enable *Bioclipse* for this, the resource schema shown in Figure 36 was implemented. Therefore, an interface *IDBResource* was created as well as a *PersistedDBResource* implementing it.

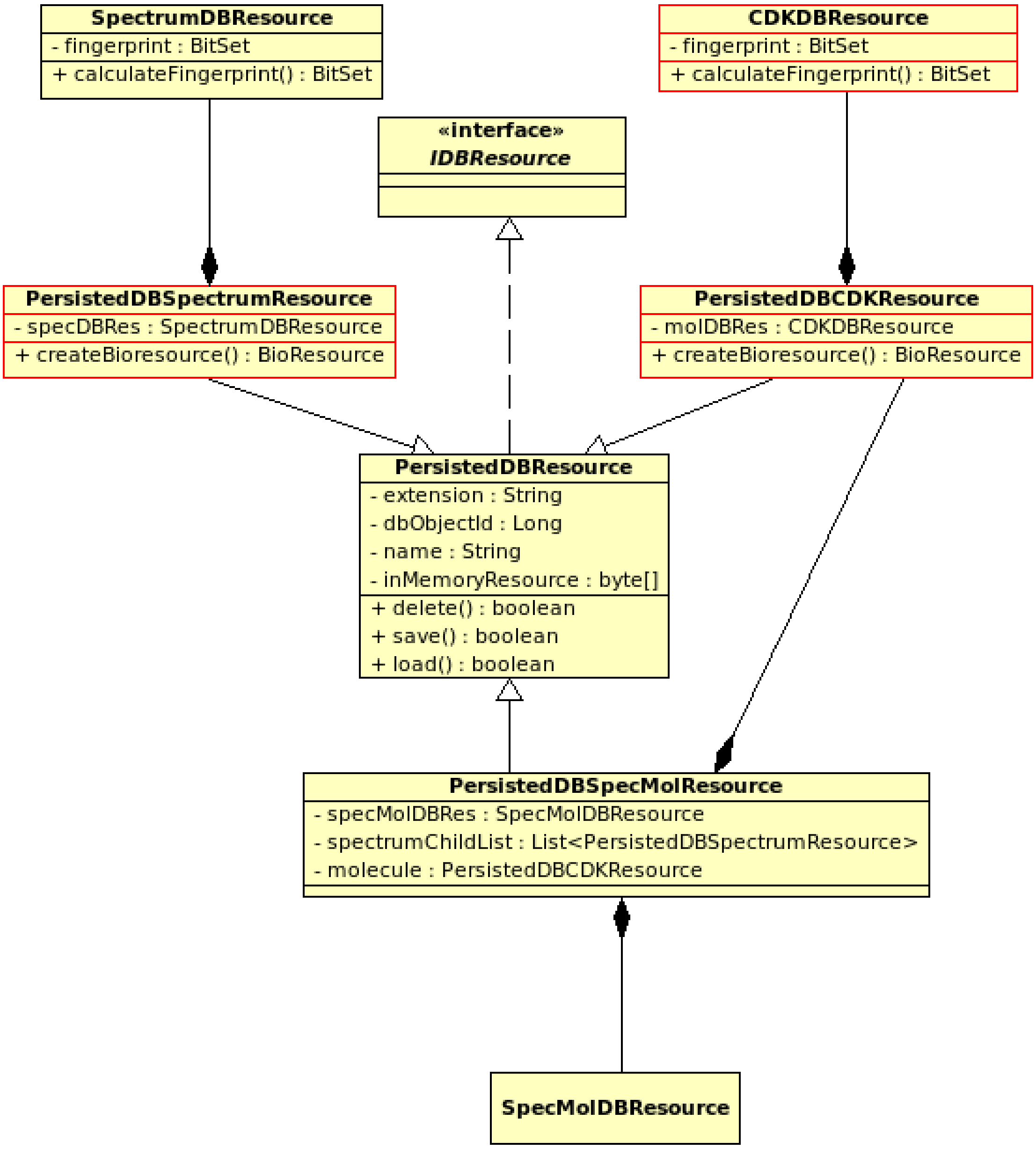

SpectrumDBResource
- fingerprint : BitSet
+ calculateFingerprint() : BitSet
CDKDBResource
- fingerprint : BitSet
+ calculateFingerprint() : BitSet
«interface»
IDBResource
PersistedDBSpectrumResource
- specDBRes : SpectrumDBResource
+ createBioresource() : BioResource
PersistedDBCDKResource
- molDBRes : CDKDBResource
+ createBioresource() : BioResource
PersistedDBResource
- extension : String
- dbObjectId : Long
- name : String
- inMemoryResource : byte[]
+ delete() : boolean
+ save() : boolean
+ load() : boolean
PersistedDBSpecMolResource
- specMolDBRes : SpecMolDBResource
- spectrumChildList : List<PersistedDBSpectrumResource>
- molecule : PersistedDBCDKResource
SpecMolDBResource

Figure 36: *UML* class diagram showing the relationships of the newly introduced resources to each other. This enhancement of the existing resource schema was necessary to support the storage of calculable properties to the database together with the resources' object.

The *PersistedDBResource* is extended by one class per resource, that should be persisted, and is actually providing the methods for saving, loading and deleting the object. Additionally, it holds all the general fields to be persisted, like the id of the resource, its name and its extension. The object itself is stored as a string representing the data content in the regarding file format, so that the resource can be parsed in the same way like its non-database persisted representative. The *PersistedDBResource* extending classes are adding the fields to the resource, that should be persisted additionally to the object. As an example, fingerprints for spectra and molecules were added as such fields, but many other variables can be imagined. These values are calculated on first import into the database and then stored together with the object. This enables the system to perform fast searches using just these variables without the need to retrieve the whole resource and parse it.

Within the Bioclipse application, persisted resources are responsible for the actual reading and writing of the underlying data to and from its source. Therefore, additional *IBioResource* implementations where added to the system for every object to be persisted within a relational database as well. Consequently, these resources care about parsing and interpretation of the saved objects and provide methods for the calculation of the enhancing variables.

The *HSQLDB* (*Hypersonic SQL Database*), a lightweight 100% *Java SQL* database engine distributed under an open source license (see Chapter 1.4.1), was used as an example for the integration of database systems. *HSQLDB* can easily be used as embedded database engine in two ways, in memory only or disk based. In the same manner any other system supported by *Hibernate* could be integrated as well. Any supported database system, connected to *Bioclipse* via the regarding extension point, is visible to the user as a new virtual folder within the resource navigator bearing the name provided via the regarding extension.

For adding new datasets to one of the connected databases, a wizard was created. The action to start this wizard can be accessed in two ways, either via the context menu of the database folder or via an entry within the database menu. The wizard allows for the selection of the database the data should be persisted into and the selection of one or multiple resources from the actual resource tree. By finishing the wizard, the selected resources file content is read, the regarding *PersistedDBResources* created, the data object copied, the related *BioResources* created, all data to be persisted calculated and the whole object saved to the relational database.

Additionally, actions for retrieving all three implemented resource types separately are included. On running one of theses actions, all regarding resources are fetched from the database, the regarding *BioResources* created and added as children to the regarding database folder. These resources can then be used in the same manner as any other resource in *Bioclipse*. To clear the database folder from formerly retrieved datasets, another action was added to the folders context menu.

6 Semantics and Dictionaries for Metabolomics Data Representation

In biosciences in general and in *Metabolomics* and systems biology in particular, large amounts of detailed chemical information is required and published. The data obtained by *Metabolomics* research is to a major percentage describing small and medium sized molecules. Therefore, methodology from chemistry can and should be adapted to describe and share this data without information loss, respectively procedures already standardised within the life sciences have to be adopted by the chemical community. For example, the requirement to deposit certain experimental data in open access repositories before publication is very common in life sciences. In contrast, chemistry has a history of closed data collections maintained and brought to the market by chemical enterprises. These data collections are often even just gained by exhaustive mining of primary literature, so that scientists pay later on for accessing the data they and their collogues already published.

Additionally, a major part of chemical data resulting from publicly funded research is never published [77] and therewith lost for the community, as publication in peer reviewed journals is still the predominant way of communicating information in scientific communities. In this context, the internet is of growing importance, as it evolved into one of the most important sources of information in general and in sciences in particular. This development was realised by the publishers too, so that nearly all scientific papers can be obtained in an electronic form at least additionally to the classic printouts, if not exclusively.

As most of these electronic media are at the moment nothing more than digital replicas of their paper-based counterparts, the advantages provided by existing web technologies are normally not exploited. This happens, although a big surplus value could be added to the data entities by e.g. linking them to each other and unambiguously defining them using controlled vocabularies, thesauri, dictionaries and ontologies. This would allow for mapping the semantic relationships between data entities and for the preservation of these within the publication process. A dictionary in this context is defined as a unique definition of terms, whereas a thesaurus is used for defining relations between these terms, and an ontology structures these relationships in a hierarchical manner.

The real proceeding within these improvements would lie in the possibility to browse, find and utilize the so published data not just by humans, but by automatic routines as well. This vision of a knowledge management framework for navigation and discovery of distributed resources on the web was described by Tim Berners-Lee and James Handler in [78] as the "Semantic Web". The *Semantic Web Activity* of the *W3 Consortium* is therefore developing new technologies facilitating the creation of machine-readable content [79].

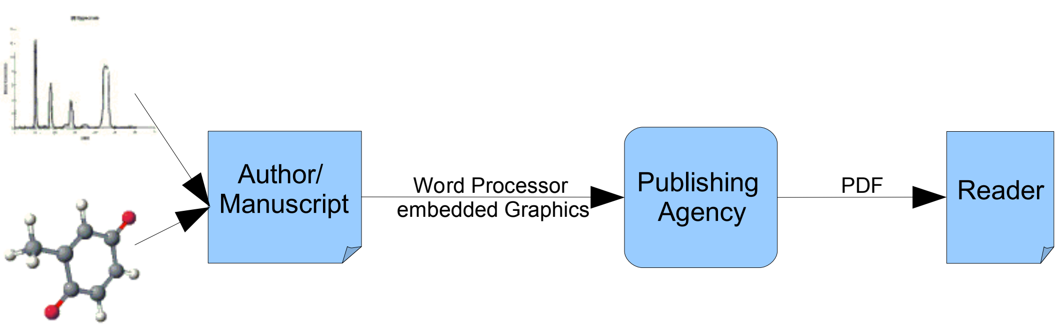


Figure 37: Schematic depiction displaying the actual publishing process.

The schematic diagrams of two different publication processes are shown in Figure 37 and Figure 38. Whereas in Figure 37 the traditional way is shown with its massive loss of information and semantics, Figure 38 visualises a possible alternative, which is supported by the tools and formats developed and defined within this thesis.

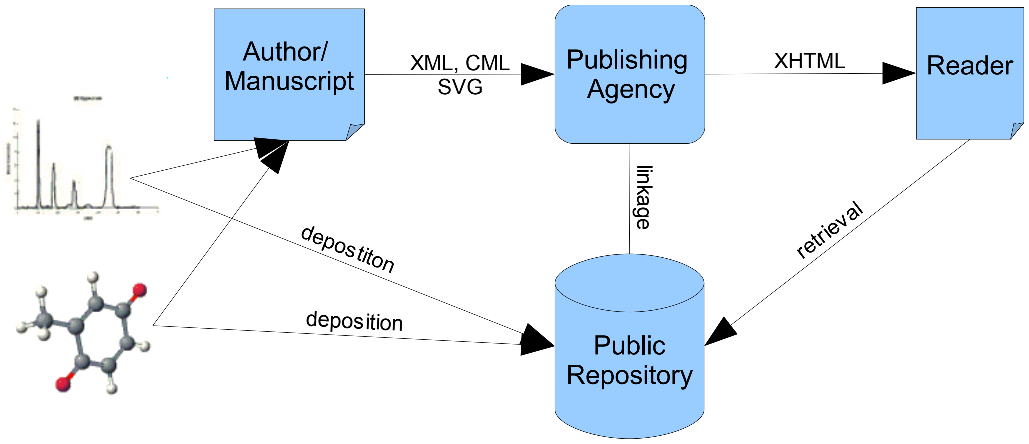


Figure 38: Enhanced publishing process pointing out the preservation of semantics together with the deposition of associated data in public repositories.

Because chemical data traditionally is validated by associated information about properties and analysis, every publication of a new compound is accompanied by measurements of these for justification. This leads to a very broad information base "available" mostly as printed text extended by graphical depictions also forming the major feedstock of the pharmaceutical industry for the development of new drugs. So, the enhancement of the publication process with semantics would not just lead to better quality and efficiency in academic research, but would improve the drug development process sustainably as well. An important prerequisite for enabling publication of data under preservation of semantics is the deposition of this data in freely accessible data repositories. These are probably best maintained by public institutions, as especially the public funding agencies would have interest in the data produced with their help, being available for future research. The so deposited data would then on the one hand have to be interlinked and on the other hand be connected to the primary publication, so that it would be possible to access all relevant data starting from any of the deposition points.

There are first approaches from publishers into this direction. The *Royal Society of Chemistry* for example, recently announced an initiative of enhancing their journal papers so that the data published can be read, indexed and intelligently searched by machines. Additionally, they are going to directly include information on molecular structures and scientific concepts into the publication or link them to available electronic databases [5].

Another major advantage for publishers and authors in publishing data in an agreed machine processable format, is the augmentation of publication quality. An automatic data validation by regarding software would lead to a simplification of the quality assurance process on both sides – publishers and authors. Additionally, open scientific software helping in authoring this type of publications is already being developed and will further evolve. Indeed, the publication in compliance with the constraints needed to realise the "Semantic Web" does not mean real extra work for neither side, it is just about accepting the new work flows. In addition to the necessity of designing the concepts underlying this new publishing process, the implementation of high quality free software products is needed. These would reduce time and money to be invested by the involved parties.

Necessary prerequisites for the realisation of this vision are the formalisation of data-representations in machine-readable and processable formats and vocabularies. Furthermore, supportive software systems that process these information without human intervention have to be developed. Agreement on certain open standards to define these formats and vocabularies enables the success of this approach by giving the community the possibility to develop applications and other technologies on top of them. The *eXtensible Markup Language* (*XML*), as a highly structured data format, that is made for being both, human and machine understandable, lends itself to form the basis for many of these emerging technologies.

In the next chapters this and the *Chemical Markup Language* (*CML*), a *XML* vocabulary for molecular data, will be introduced and a special vocabulary for spectral data and related dictionaries for the definition of spectral meta data developed within this thesis will be presented and explained in detail.

6.1 The Extensible Markup Language (XML)

XML is a simple, general-purpose text format derived from *SGML* (*Standard Generalized Markup Language* - *ISO 8879*) for modelling data in a tree-like structure. Because every *XML* document has to conform to a set of standardised rules defined and maintained by the *W3C* (*World Wide Web Consortium*), *XML* encoded documents are platform independent and highly portable [80].

It was created by the *World Wide Web Consortium* to overcome the limitations of *HTML*, the *Hypertext Markup Language*. Like *HTML*, *XML* is based on *SGML* (*Standard Generalized Markup Language*). Although *SGML* has been used in the publishing industry for decades, its perceived complexity intimidated many people from other areas from using it.

In contrast to *HTML*, that has a strong focus on the presentation of the data, *XML* focusses on the content, its structure and its semantics. Therefore, it is possible to have several different ways of presenting the same *XML* enclosed textual information by using different so called transformations for generating the presentational layer.

In the beginning, *XML* was designed to be a web-aware container for data managed by legacy systems, but it has evolved until today to a more general way to model components of information systems [81]. In fact, it is a meta-language for describing markup languages, as the set of elements and attributes to be used and the structural relationship between them are not fixed, but are to be defined by the user [82].

The format is flexible enough to be used for domains, as diverse as web sites, electronic data exchange, vector graphics, mathematical equations, object serialisation, remote procedure calls, voice mail systems, scientific information and many more. This is realised by the definition of a set of human-readable tags used to markup the data and therewith identifying structures within the document [82].

The *W3C* specification just defines some general rules, like elements must have a starting and ending tag and each attribute must have a single value. The actual definition of the elements and attributes is up to the user/developer.

XML has a lot of advantages in comparison to other data formats, it is:

- Simultaneously human and machine-readable
- It is to some extend self documenting

- It is platform independent
- It is based on international standards
- By its hierarchical structure most data structures can be represented

Therewith, XML can be used to encode data to be used by very different applications without the need to communicate a complex set of rules on how to parse this data into meaningful information.

XML documents following all the notational and structural rules for *XML* are called well-formed. Programs processing *XML* should check any document for well-formedness and reject input that does not follow the defined rules. So being well-formed is equivalent to a document being parseable or not.

An element within a XML document is defined as the content being surrounded by a start and a end tag plus the tags themselves. Every element might contain several attributes, which are providing additional information for the regarding element. Attributes start with an identifying string followed by an equals sign and the assigned value in quotations.

The structure of the elements within a *XML* document and their attributes may be defined by one or multiple *Document Type Definitions* (*DTD*). A *DTD* contains a set of rules controlling how the elements are ordered, nested and combined. *DTD's* are not fixed to one *XML* document, but can be reused for other documents as well. By using the same *DTD* for a series of documents, applications are given the possibility to interpret the information from the *DTD*, analyse the *XML* document, and to validate its data before presenting it to the user or performing any processing steps. Other techniques for controlling/defining the structure of *XML* documents are *XML Schema, XML Namespaces*.

XML Schema is the successor of *DTD* and first of all differs from *DTD* by being completely *XML* based. Additionally, *XML Schema* extends the old *DTD* based system as well with respect to the content as with respect to structural features. This is realised by the specification of new data types and new elements together with their attributes. Any concrete implementation of a *XML Schema* is called a *XSD* (*XML Schema Definition*) [83][84][85].

By using *XML Namespaces* the uniqueness of elements and attributes can be assured. Elements and attributes defined in more than one vocabulary can be used within one *XML* document. If each of these vocabularies gets another namespace assigned, ambiguity between identical entries can be resolved.

A namespace in a *XML* instance is a declared pointing of an eventually existing prefix to a *URI* (*Uniform Resource Identifier*). This *URI* is not treated as a web address, but as a simple string by the *XML* parser. A *XML* namespace does not require that its vocabulary is defined, though it is fairly common practice to place either a *DTD* or a *XML Schema* defining the precise data structure at the location of the namespace's *URI* [86].

With the help of the *XML Linking Language* (*XLink*) it is possible to interconnect *XML* data elements, even from within different *XML* documents, to each other. In contrast to the hyper links used in *HTML*, *XLink* is not unidirectional, but allows for bi- and multi directional references [87].

The *XML Pointer* (*XPointer*) technology extends *XLink* to support addressing into the internal structures of *XML* documents. *XPointer* builds upon the *XML Path Language* (*XPath*) to reference to elements, selections, character strings and other parts of *XML* documents, without the need to have a jump point (e.g. *ID* attribute) defined at that position [88].

XPath was mainly developed for directly addressing parts of *XML* documents. In support of this primary purpose, it also provides basic facilities for manipulation of strings, numbers and booleans. It is the result of an effort to provide a common syntax and semantics for functionality shared between *XSL Transformations* and *XPointer* [89].

The *Extensible Stylesheet Language* (*XSL*) is in fact two different languages used to format and transform *XML* data. The two components of *XSL* are

- *XSL-FO (XSL Formatting Objects):* used for formatting *XML* data
- *XSLT (XSL Transformation):* used for the transformation of one *XML* dialect into another format

These two components are often combined together with the already mentioned *XPath* technology. The sub-languages can be used together as *XSL* or independently [90]. *XSLT* is for example used for the transformation of *XML* based datasets to an *HTML* based presentation, giving the possibility to combine the storage of data in a semantically rich format and the appealing and universal presentation via *HTML* browsers.

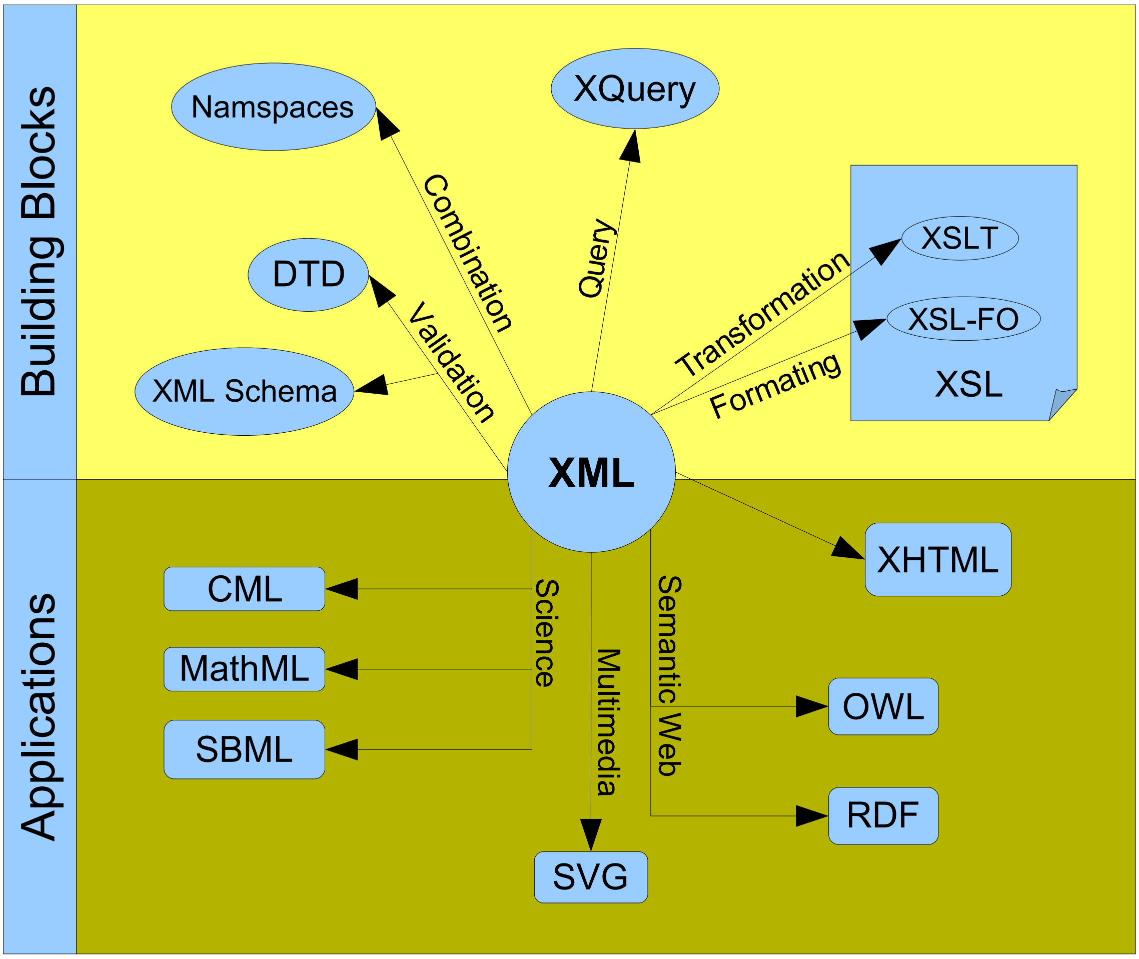


Figure 39: Diagram showing the connection of XML to other relevant vocabularies and applications.

The large amount of information being stored, exchanged and presented using *XML* itself, or *XML* compliant dialects, led to the need to intelligently query these data sources. The capabilities of *XPath* turned out to be to restricted for fulfilling this, so the *XML Query Language* (*XQuery*) was implemented, using *XPath* and *XML Schema* for the data model and its function library, to meet this requirements. It is a query language designed to query large collections of *XML* data by using a *SQL* similar syntax extended by a set of features known from modern programming languages [91].

In the natural sciences, and especially in chemo- and bioinformatics, exchanging, storing and analysing large amounts of data is of major importance. This is why the scientific community very early realised the benefits it could obtain from using/adopting *XML*. This led to the development of a huge variety of different *XML* vocabularies for many different scientific fields [92]. Some examples are:

- **BSML (Bioinformatic Sequence Markup Language):** encodes biological sequence information and includes graphical representations of biologically meaningful objects such as sequences, genes, electrophoresis gels, and multiple alignments [93].
- **BioML (Biopolymer Markup Language):** designed to be used for the annotation of biopolymer sequence information. *BioML* allows the full specification of all experimental information known about molecular entities composed of biopolymers, for example, proteins and genes [94].
- **SBML (Systems Biology Markup Language):** a computer-readable format for representing models of biochemical reaction networks. *SBML* is applicable to metabolic networks, cell-signalling pathways, regulatory networks, and many others [95][12].
- **PDBML (Protein Data Bank Markup Language):** the representation of archival macromolecular structure data in *XML* [96].
- **AnIML (Analytical Information Markup Language):** the *XML* standard for analytical chemistry data [97].
- **CML (Chemical Markup Language):** a flexible text format designed to facilitate the interchange and deposition of chemical information (see Chapter 6.2) [65][71][98].

Additionally most of the major biological and chemical database systems are providing an *XML* output of their data [99].

Recapitulating, the major advantages of using *XML* in sciences are the capability of storing data in a semantically rich manner, the extendibility of any *XML* based format without loosing the information already encoded using that format, the possibility to parse any *XML* compliant file using standard *XML* parsers, the better searchability and the easy exchange of *XML* based data [100].

6.2 The Chemical Markup Language (CML)

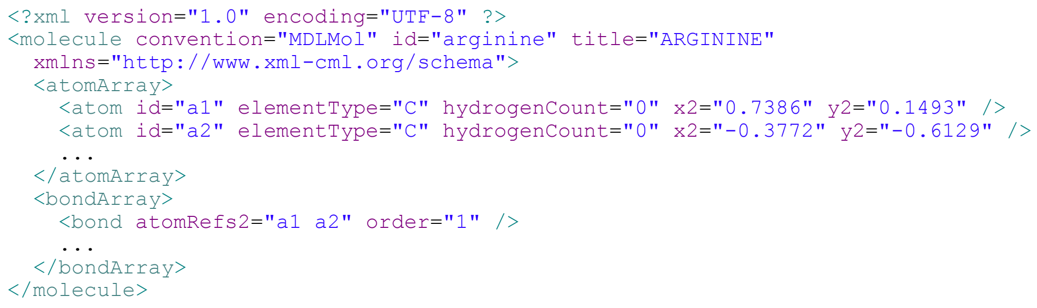
The *Chemical Markup Language* is the result of the collaborative approach to tackle some of the problems existing within the scientific community regarding the exchange of chemical information via the internet and other networks. It was developed to hold chemical entities (molecules, crystallographic data, reactions and further chemical concepts) using an *XML* vocabulary and for the first time offers a universal, platform and application independent format for storing and exchanging chemical information [65].

CML deliberately does not cover all chemistry but concentrates on "molecules" and data describing molecules or their properties.

element	short description
<angle>	element represents a valence angle and may be used to construct a molecule from internal coordinates or z-matrix
<atom>	represents an atom in a molecule
<bond>	element represents a bond in a molecule
<crystal>	element represents a unit cell
<electron>	element represents a electron
<feature>	element represents a feature in a biomolecule. It is mainly to support sequence and structure files and will probably be mainly textual
<formula>	element represents the atom count in a molecule. It can be hierarchical (i.e. a formula can contain sub-formulas recursively)
<molecule>	represents a "molecule" as a group of atoms and bonds. It can be hierarchical (i.e. a molecule can contain sub-molecules recursively)

Table 1: Overview of the most important molecular elements contained in the CML core definition.

An overview of the most important molecular elements of *CMLCore* is displayed in Table 1 (extracted from the *CML FAQ*s [101]). The modularized structure of *CML2* is described in more detail later in this chapter (see page 94).



```
<?xml version="1.0" encoding="UTF-8" ?>
<molecule convention="MDLMol" id="arginine" title="ARGININE"
  xmlns="http://www.xml-cml.org/schema">
  <atomArray>
    <atom id="a1" elementType="C" hydrogenCount="0" x2="0.7386" y2="0.1493" />
    <atom id="a2" elementType="C" hydrogenCount="0" x2="-0.3772" y2="-0.6129" />
    ...
  </atomArray>
  <bondArray>
    <bond atomRefs2="a1 a2" order="1" />
    ...
  </bondArray>
</molecule>
```

Figure 40: Image displaying a section of a *CML* document encoding the amino acid arginine.

The example in Figure 40 shows a *CML* document representing the amino acid arginine. The molecule is containing an array of atoms, formed by multiple atom elements and an array of bonds, formed by multiple bond elements, that link via the *atomRefs* attribute to the regarding atoms. With this attribute a listing of atom ids is stored as strings. This way of using references is the standard procedure of linking entities within *CML* to each other.

CML was one of the first specialist markup languages and therefore is both mature and well established in the chemistry community.

Because *CML* is, as any *XML* compliant language, a plain text file format and because it is capable of holding extremely complex information structures, the chemical markup language is especially well suited to be used for long term archival of chemical data. Even if the regarding *XMLSchema* or *DTD* is not available (any more), based on the intuitive labelling of the elements it will still be possible to extract most of the information from the documents.

Furthermore, it is thought to be of major help within the process of publishing scientific data, by giving the submitting person the possibility to directly associate the data sources underlining their conclusions to the textual representation within the same document.

Because the authors can provide all the important information in one go, and the publisher just converts the relevant information to e.g. tables, graphics and diagrams this would as well shorten and improve the publishing process strongly. Thereby, the eventuality, that the (at the moment not even always attached) supplementary material will later not be available any more with the publication itself would be eliminated. This would give anybody the chance to better understand and/or recapitulate the published facts.

Especially for data published on the Internet, in open data repositories, but as well for the journal based publishing process and in house databases, a confirmation of the validity of the data is of major importance. For this purpose G. Gkoutos et al. [102] described a mechanism for the certification of chemical data and meta data using digital signatures, based on the established X.509 certification technology [103]. This includes not only the certification of data by persons or organisations, but highlights as well how a document or a part of a document could be signed by an application. So, e.g. a *CML* encoded molecule generated by a molecular calculation application could be signed by this program automatically.

CML is a project still under development with its functionality being extended whenever the need for new concepts comes up. By being based on a flexible framework and trying to reuse existing concepts and protocols defined by the *W3C* wherever possible, the project is well prepared for ongoing evolution.

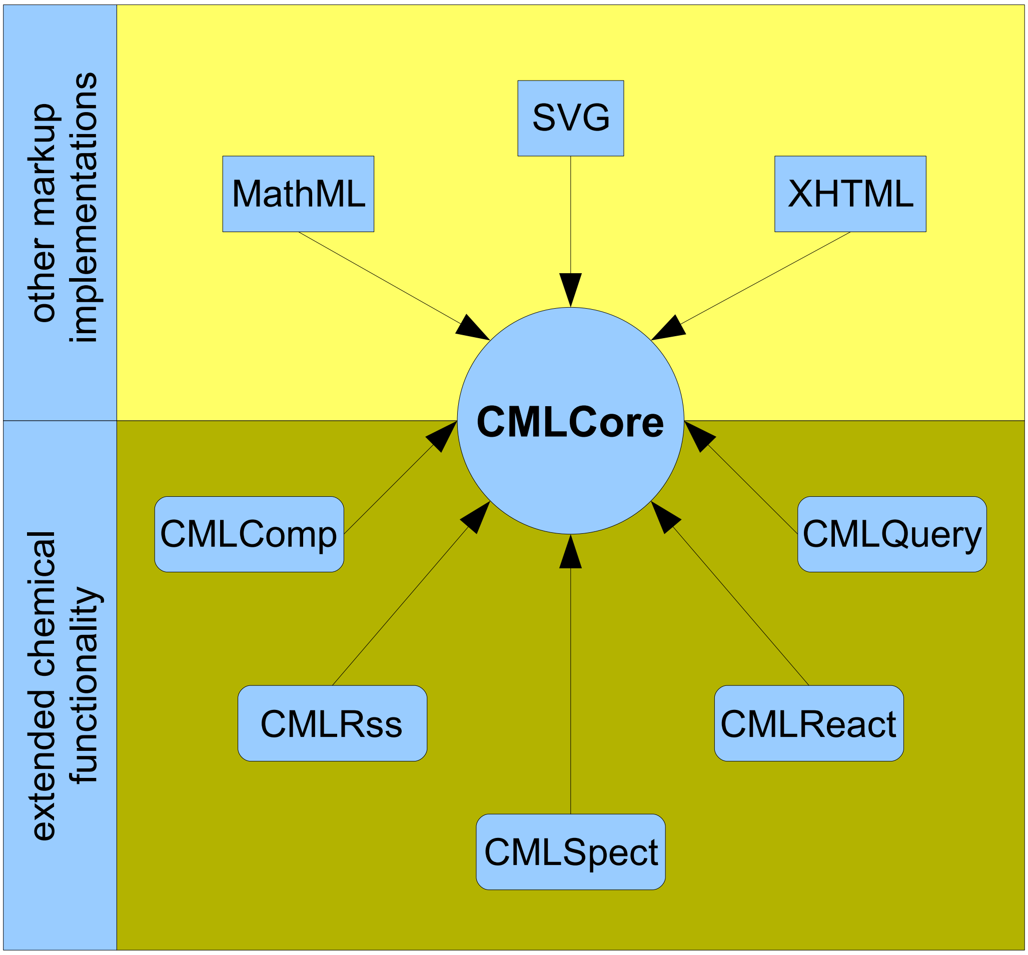


Figure 41: Diagram showing the CML components and some further XML languages, that integrate well with CML. The CML components are defined as follows: CMLCore – molecular and related information, CMLReact – chemical reactions, CMLSpect – spectral information (see 3.2.1), CMLComp – computational chemistry, CMLRSS – an XML based RSS carrier for chemical data and CMLQuery – general query language for chemistry. MathML is the Mathematical Markup Language, for describing mathematics, SVG stands for Scalable Vector Graphics, a standard for describing 2D vector graphics and STMML (Scientific-Technical-Medical Markup Language) is a markup language for scientific, technical and medical publishing (image adapted from [Murray-Rust2003]).

The *Chemical Markup Language* in version 2 is designed in a modular way, by a core component, providing the general framework for encoding molecular data and some further components extending this framework to support advanced concepts like reactions and properties. Additionally, it is possible to seamlessly integrate information encoded in any other *XML* dialect by ensuring the uniqueness of element names via namespacing.

In Figure 41 the different components forming *CML* and their relationship to some extending markup languages is shown.

The following list gives a short overview on the different components and a short description of them [98]:

- *CMLCore:* used to encode molecular information and related meta data
- *CMLReact:* for representation of chemical reactions [104]
- *CMLSpect:* holds spectral information and related meta data (see Chapter 6.3)
- *CMLRSS:* an *XML*-based *RSS* carrier for chemical data [56]
- *CMLComp:* for encoding computational experiments
- *CMLQuery:* adds query functionality to *CML*

All non-chemical concepts, like for numeric data, data structures (arrays and matrices), scientific units, etc., are contained in additional markup dialect - the *Scientific*, *Technical* and *Medical Markup Language* (*STMML*) [105][106]. Since version 2 *CML* is using *XMLSchema* instead of a *DTD* for entity definition. This allows *CML* to validate both the structure of documents containing *CML* components and the data types of these components by using generic processing tools.

CML is specified more formally by a *Document Object Model* (*CMLDOM*) that contains interfaces and methods constructed using a collection of *Java* classes. The *CMLDOM API* (*API = Application Programming Interface*) is to be used by software developers/providers for the programmatic creation, manipulation and export/import of chemical information using the *Chemical Markup Language* [107][71]. This *Java* implementation of the *CMLDOM* is based on *XOM*, a new open-source, tree-based *API* for processing *XML*, that aims at correctness, simplicity and performance [108].

The *Chemical Markup Language* has already been used in the chemical community to manage very diverse chemical documents and information including:

- Spectra
- Organic molecules
- Different types of publishing

- Inorganic crystallography
- Regulatory processes
- In databases
- Macromolecular sequences and structures

Therewith, *CML* is perfectly fitted to handle chemical information under preservation of its semantics.

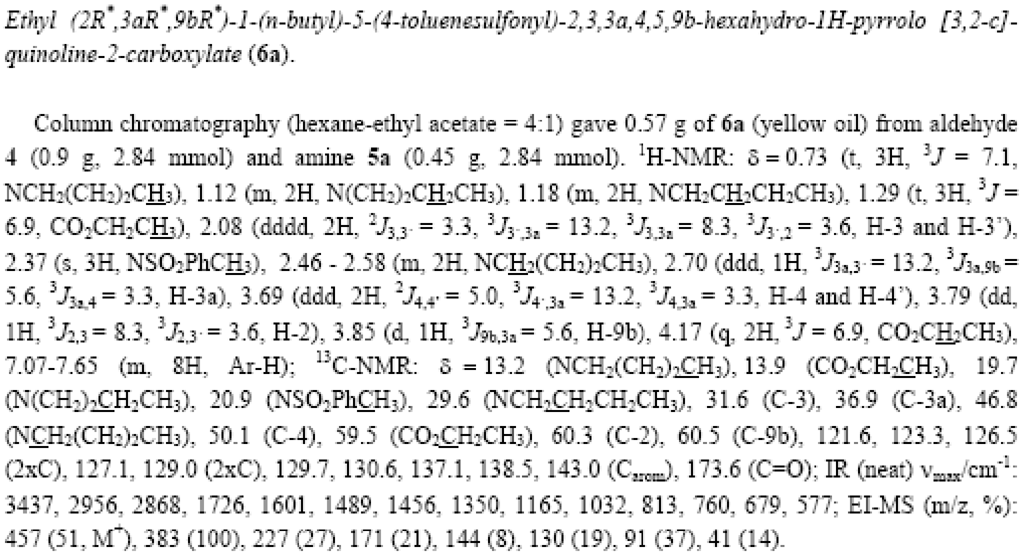
6.3 The CMLSpect Vocabulary for Spectral Data

The *CMLSpect* definition was developed during this thesis in collaboration with the group of Dr. Murray Rust from the Unilever Centre for Molecular Informatics, Cambridge, UK. *CMLSpect* is an enhancement of the formerly described *Chemical Markup Language* for encoding spectroscopic data (see Chapter 1.3).

CMLSpect was designed to meet several aims. First of all, it is used to encode general spectral data. Beside this, it is capable of storing annotations for this primary data, like e.g. peak shapes, the coupling of multiple peaks in a *NMR* spectrum and other information important for, or resulting from spectrum interpretation. Additionally, it is able to hold associated meta data, like information about the experiment itself, the molecule measured and the machine used for the measurement. Furthermore, the possibility to interconnect spectral data elements with molecular *CML* data elements is implemented.

Therewith, *CMLSpect* is able to hold spectral data recorded in the laboratory as well as data resulting from simulations. As spectroscopic methods are broadly applied and it is effectively mandatory to report their results with the publication of novel compounds, there exist a large amount of this data hidden in printed publications. Together with the other *CML* dialects *CMLSpect* allows to encode the information contained in analytical blocks (see Figure 42 for an example) found in these publications. The extraction of this information using text mining approaches would give access to the chemical data published in the last 50 years to be used e.g. for structure dereplication and elucidation. However, this is a very demanding and difficult task, as the format of these analytical blocks is not uniformly defined, which leads to a big variety of slightly different appearances, sometimes even within the same journal.

Nevertheless, there are tools being developed to enable this “chemical archaeology”. One example is a tool called *OSCAR* (*Open Source Chemistry Analysis Routines*) [109] [110], which was integrated into the *Bioclipse* framework in cooperation with the author. *OSCAR* is performing an automatic regular expression based semantic annotation of chemical publications. It extracts structures, and spectra, displays *InChIs* (*IUPAC International Chemical Identifier*) and *SMILEs*, detects chemical names and creates structures from them, if they are *IUPAC* compliant, from publications. Therein, it reaches an average rate of correctness of ~ 50%. The *Bioclipse* integrated variant is capable of the same, just that additionally, the identified chemical entities are parsed into the regarding resources and are placed into the central navigator, whereas all other detected categories can be displayed as a document with marked up regions. Nevertheless, for the future it is probable, and supported by our developments, that published data will be stored under preservation of the semantics in some type of repository (either institutional or centralised) with linkage to the regarding publications and *vice versa.* This will allow for easy retrieval of the analytical data even without human intervention.



Ethyl (2R*,3aR*,9bR*)-1-(n-butyl)-5-(4-toluenesulfonyl)-2,3,3a,4,5,9b-hexahydro-1H-pyrrolo [3,2-c]-quinoline-2-carboxylate (**6a**).

Column chromatography (hexane-ethyl acetate = 4:1) gave 0.57 g of **6a** (yellow oil) from aldehyde **4** (0.9 g, 2.84 mmol) and amine **5a** (0.45 g, 2.84 mmol). ^{1}H-NMR: $\delta = 0.73$ (t, 3H, ^3J = 7.1, $NCH_2(CH_2)_2C\underline{H}_3$), 1.12 (m, 2H, $N(CH_2)_2C\underline{H}_2CH_3$), 1.18 (m, 2H, $NCH_2C\underline{H}_2CH_2CH_3$), 1.29 (t, 3H, ^3J = 6.9, $CO_2CH_2C\underline{H}_3$), 2.08 (dddd, 2H, $^2J_{3,3'}$ = 3.3, $^3J_{3',3a}$ = 13.2, $^3J_{3,3a}$ = 8.3, $^3J_{3',2}$ = 3.6, H-3 and H-3'), 2.37 (s, 3H, $NSO_2PhC\underline{H}_3$), 2.46 - 2.58 (m, 2H, $NC\underline{H}_2(CH_2)_2CH_3$), 2.70 (ddd, 1H, $^3J_{3a,3'}$ = 13.2, $^3J_{3a,9b}$ = 5.6, $^3J_{3a,4}$ = 3.3, H-3a), 3.69 (ddd, 2H, $^2J_{4,4'}$ = 5.0, $^3J_{4',3a}$ = 13.2, $^3J_{4,3a}$ = 3.3, H-4 and H-4'), 3.79 (dd, 1H, $^3J_{2,3}$ = 8.3, $^3J_{2,3'}$ = 3.6, H-2), 3.85 (d, 1H, $^3J_{9b,3a}$ = 5.6, H-9b), 4.17 (q, 2H, ^3J = 6.9, $CO_2C\underline{H}_2CH_3$), 7.07-7.65 (m, 8H, Ar-H); ^{13}C-NMR: δ = 13.2 ($NCH_2(CH_2)_2\underline{C}H_3$), 13.9 ($CO_2CH_2\underline{C}H_3$), 19.7 ($N(CH_2)_2\underline{C}H_2CH_3$), 20.9 ($NSO_2Ph\underline{C}H_3$), 29.6 ($NCH_2\underline{C}H_2CH_2CH_3$), 31.6 (C-3), 36.9 (C-3a), 46.8 ($N\underline{C}H_2(CH_2)_2CH_3$), 50.1 (C-4), 59.5 ($CO_2\underline{C}H_2CH_3$), 60.3 (C-2), 60.5 (C-9b), 121.6, 123.3, 126.5 (2xC), 127.1, 129.0 (2xC), 129.7, 130.6, 137.1, 138.5, 143.0 (C_{arom}), 173.6 (C=O); IR (neat) ν_{max}/cm^{-1}: 3437, 2956, 2868, 1726, 1601, 1489, 1456, 1350, 1165, 1032, 813, 760, 679, 577; EI-MS (m/z, %): 457 (51, M$^+$), 383 (100), 227 (27), 171 (21), 144 (8), 130 (19), 91 (37), 41 (14).

Figure 42: Depiction displaying a typical analytical block from a paper published in a journal of synthetic organic chemistry with information about various analytical experiments. Data for a column chromatography, 1H-NMR, 13C-NMR, IR and EI-MS experiments is shown. (from Molecules 2007, 12, 49-59: Microwave Assisted Synthesis of Substituted Hexahydropyrrolo[3,2-c]quinolines).

Because of the overall facilitation, *CMLSpect* and *CML* in general mean to the publishing process and the availability of semantically meaningful chemical data, we expect most vendors of spectrometers to adapt to these standards. This will presumably lead to algorithms to convert most vendor formats to *CML* in the near future. These will be available either as separate software libraries or better by providing *CML* exports of the measured data directly. I do not see a possibility to implement conversion tools for all vendor formats on our side, as they are mostly closed source and it would mean to much investment of time and money to refactor these data encodings. As most spectrometers already provide exports to the *JCAMP-DX* format, and our methods are capable to convert this to *CML,* there already exists a possibility to import spectrometer generated data into the systems developed within this thesis.

At the moment of writing, *CMLSpect* is capable of capturing the three most common occurrences of spectra:

- Continuous spectra $(y=f(x))$ (6)
- Discrete or peak spectra $((x_{1,} y_1)\ (x_n, y_n))$ (7)
- 2-dimensional spectra $(z=f(x, y))$ (8)

By addressing all spectral data as representations of these types of functions the design ensures the flexibility needed to handle data from most if not all currently available spectroscopic methods and hopefully is prepared for upcoming new technologies. Anyway, if necessary it is without any difficulty possible to extend the actual *CMLSpect* schema to meet future demands. Additionally, as spectroscopists and chemists use many different ontologies, that sometimes are just irreconcilable, *CMLSpect* does not define a fixed vocabulary for adding annotations, but provides the framework for precisely defining them.

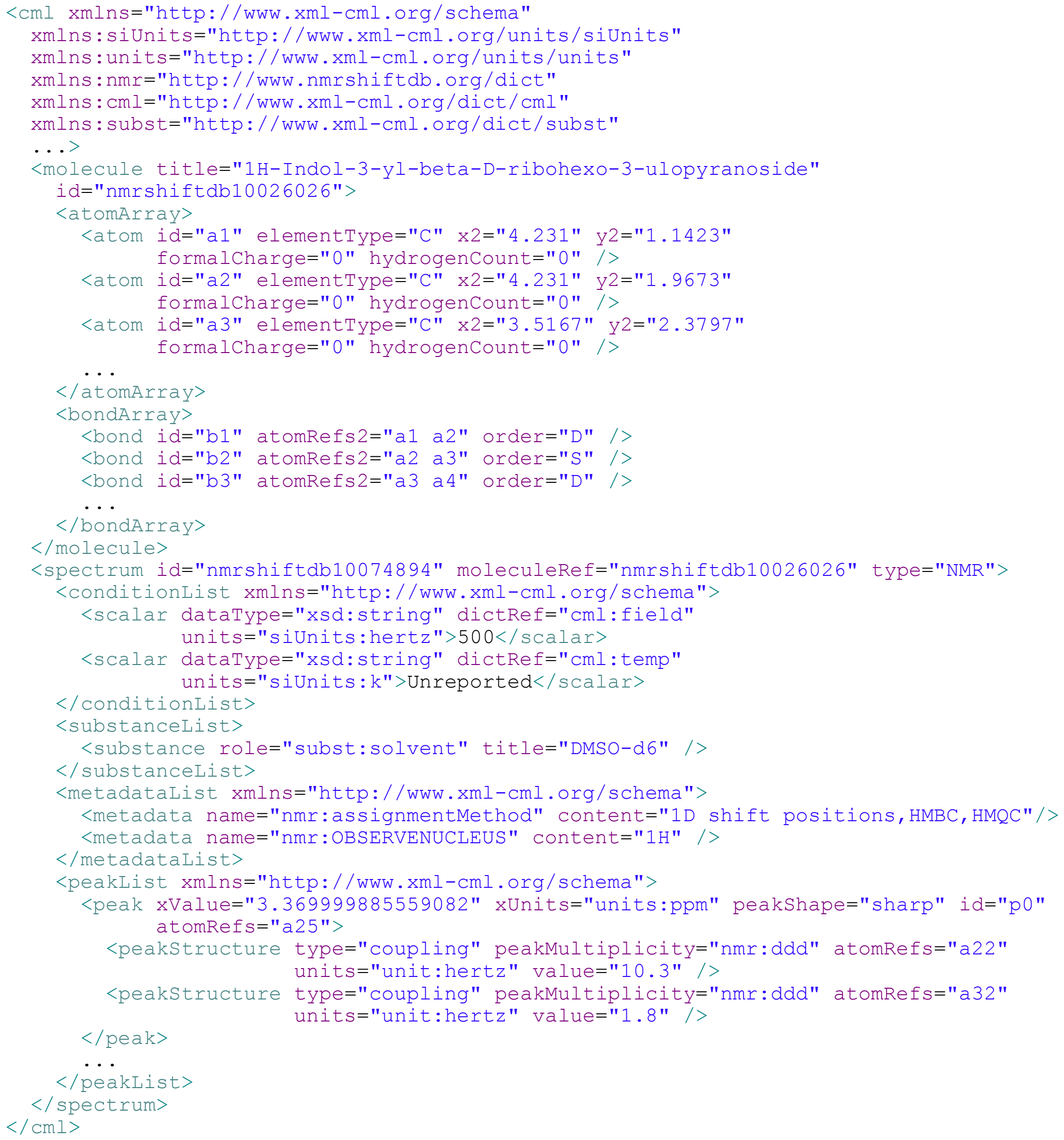
For *CMLSpect,* there were some specific elements been defined, which again are to be combined with the existing *CML* elements. Table 2 gives a short overview on these elements and their semantics. An example for a *CML* encoded spectrum is given in Figure 43 (see Chapter 9 for further examples). This example shows parts of a *NMRShiftDB* exported *CML* file containing a *^{1}H-NMR* spectrum and a molecule. Additional *CML* examples clarifying the usage and meaning of the described elements are shown in Chapter 9.

element	short description
spectrum List	specific container for holding <spectrum> elements
spectrum	the root element for every contained spectrum identified by an “id” attribute. Its type can be set using the “type” attribute.
peakList	container for holding <peak> elements
peak	a container for the real peak values
xValue	holds the actual x-value for this peak
yValue	holds the actual y-value for this peak
peakStructure	element containing details & annotations about the peak
spectrumData	top level container for continuous spectral data
xaxis	element holding the x-values in an <array> or a <matrix>
yaxis	element holding the y-values in an <array> or a <matrix>
peakGroup	enfolds several peaks having a chemical relationship

Table 2: The spectrum specific CML elements.

<spectrum> is the top level element for spectral data. Within this element all other elements describing the spectrum directly are contained, whereas indirect descriptions can be added

by using the *CML* referencing methodology (see Chapter 6.2 for more information on referencing). If more than one spectrum or multiple snippets of spectra have to be represented, a <spectrumList> containing multiple <spectrum> entries is to be used.



```
<cml xmlns="http://www.xml-cml.org/schema"
  xmlns:siUnits="http://www.xml-cml.org/units/siUnits"
  xmlns:units="http://www.xml-cml.org/units/units"
  xmlns:nmr="http://www.nmrshiftdb.org/dict"
  xmlns:cml="http://www.xml-cml.org/dict/cml"
  xmlns:subst="http://www.xml-cml.org/dict/subst"
  ...>
  <molecule title="1H-Indol-3-yl-beta-D-ribohexo-3-ulopyranoside"
    id="nmrshiftdb10026026">
    <atomArray>
      <atom id="a1" elementType="C" x2="4.231" y2="1.1423"
            formalCharge="0" hydrogenCount="0" />
      <atom id="a2" elementType="C" x2="4.231" y2="1.9673"
            formalCharge="0" hydrogenCount="0" />
      <atom id="a3" elementType="C" x2="3.5167" y2="2.3797"
            formalCharge="0" hydrogenCount="0" />
      ...
    </atomArray>
    <bondArray>
      <bond id="b1" atomRefs2="a1 a2" order="D" />
      <bond id="b2" atomRefs2="a2 a3" order="S" />
      <bond id="b3" atomRefs2="a3 a4" order="D" />
      ...
    </bondArray>
  </molecule>
  <spectrum id="nmrshiftdb10074894" moleculeRef="nmrshiftdb10026026" type="NMR">
    <conditionList xmlns="http://www.xml-cml.org/schema">
      <scalar dataType="xsd:string" dictRef="cml:field"
              units="siUnits:hertz">500</scalar>
      <scalar dataType="xsd:string" dictRef="cml:temp"
              units="siUnits:k">Unreported</scalar>
    </conditionList>
    <substanceList>
      <substance role="subst:solvent" title="DMSO-d6" />
    </substanceList>
    <metadataList xmlns="http://www.xml-cml.org/schema">
      <metadata name="nmr:assignmentMethod" content="1D shift positions,HMBC,HMQC"/>
      <metadata name="nmr:OBSERVENUCLEUS" content="1H" />
    </metadataList>
    <peakList xmlns="http://www.xml-cml.org/schema">
      <peak xValue="3.369999885559082" xUnits="units:ppm" peakShape="sharp" id="p0"
            atomRefs="a25">
        <peakStructure type="coupling" peakMultiplicity="nmr:ddd" atomRefs="a22"
                       units="unit:hertz" value="10.3" />
        <peakStructure type="coupling" peakMultiplicity="nmr:ddd" atomRefs="a32"
                       units="unit:hertz" value="1.8" />
      </peak>
      ...
    </peakList>
  </spectrum>
</cml>
```

Figure 43: Parts of a *NMRShiftDB* exported *CML* file containing a ^{1}*H-NMR* peak spectrum spectrum and a molecule. The usage of the most important spectrum relevant elements and their attributes is demonstrated (additional more complete example files can be found in Chapter 9).

A <spectrum> has two important attributes - id and type. The id should hold an unique identifier for this spectrum, whereas type defines the type of the spectrum and can currently be set to *NMR*, *massSpectrum*, *UV/VIS* and infrared. Dependent on the occurrence of the spectrum (if it is continuous or discrete) the <spectrum> contains <spectrumData> or a

<peakList>. If both forms of information are available a <spectrum> might contain both of them as well. There could multiple <peakList>s be defined within one spectrum. These could e.g. encode for the results of different peak picking methods applied to the same continuous data. This shows once more the already mentioned attempt to hold the definition of the vocabulary as flexible as possible.

A <spectrumData> element holds a <xaxis> and a <yaxis>. Inside these, either <array> or <matrix> are used to store the actual values, depending on the dimensionality of the data. Both are representations of primitive data types and cannot contain any complex objects, but hold the actual data elements separated by a delimiter. This delimiter is per default a "white space", if any other delimiter should be used, it has to be defined via the delimiter attribute. If for an <array> a starting point, an endpoint and a size are defined via the regarding attributes, the array itself will be generated implicitly. Together with <scalar> these two elements are the primary methods for communicating scientific data not having special elements within *CML*. To add semantics to these elements references to dictionaries are used via the dictRef attribute (see Chapter 5.2.1 and Figure 19 for further information on dictionaries). This approach can and should be used to extend other existing *CML* elements as well. Another important attribute of this primitive elements is the dataType, which helps software to decide how to process the contained data by defining the data type to be expected.

In contrast to this a <peakList> is formed by multiple <peaks> . The actual numeric values are assigned to those <peak>s by the attributes xValue and yValue. Other important attributes of the <peak> element are atomRefs, bondRefs and moleculeRefs, via these an assignment of peaks to structural features is possible. A <peakGroup> is used to combine several <peak>s showing a chemical relationship, the formerly described attributes can be used in this context as well.

To annotate a spectrum it most often is reduced to its peaks and other subsidiary features (e.g. the peaks shape, or its size). So in *CMLSpect* a peak is just defined as a region of a spectrum identified by the author which he/she wants to comment on. In many cases peaks are a (lossy) way of communicating a spectrum in form of its peak fingerprint. This was a very common way to represent spectral data in times where computer memory was still heavily limited. In nowadays it can still be used as additional information beside the full spectral data set for performing fast searches and comparisons. However, as it can be very useful to mark and annotate regions of a spectrum, *CMLSpect* provides some additional elements and attributes for this.

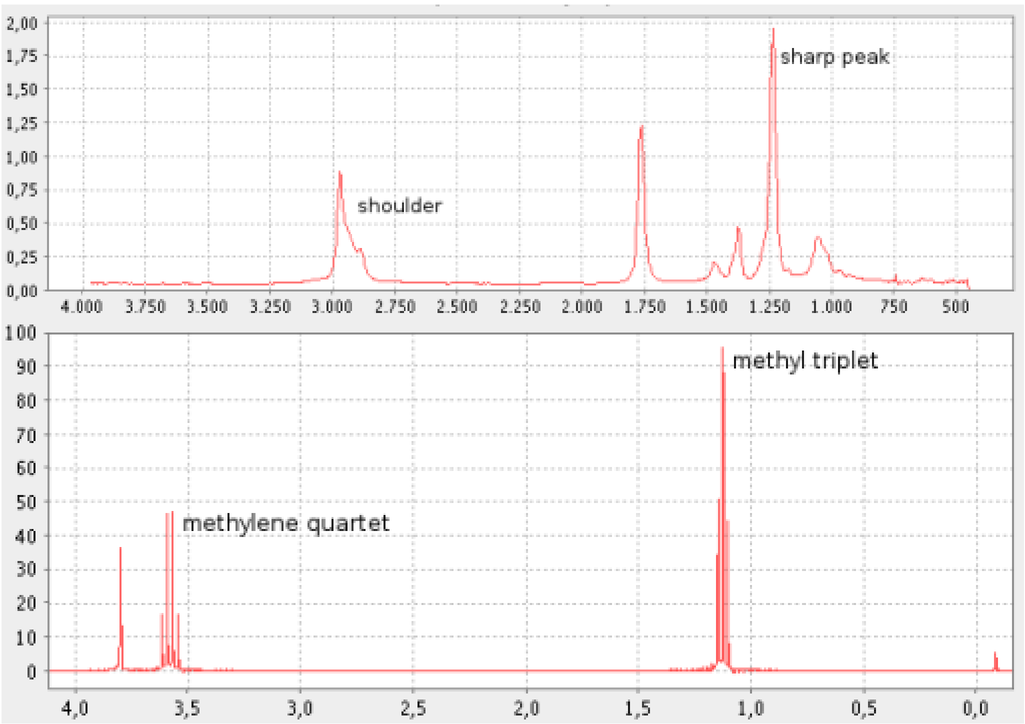


Figure 44: Two sample spectra visualising the meanings of peakStructure and peakShape. The upper one is a IR spectrum of 1-butanol-3-methyl-acetate showing two different peak shapes, a sharp peak and a shoulder. The second one is a 1H NMR spectrum of ethanol showing two different couplings, for the methyl triplet and the methylene quartet.

Thus, a peak can have an attribute peakMultiplicity, which for example can define a triplet in a *NMR* spectrum (see Figure 44 lower spectrum). The element <peakStructure> is thought to give details about the peak like e.g. *NMR* couplings. Therefore, its attribute type has to be set to “coupling”, the attribute atomRefs defines the atom to which the peak couples and with the help of the attributes value and units the coupling constant would be stored. By the attribute peakShape of <peak> the form of a peak can be defined, e.g. as being “sharp” or showing a “shoulder” (see Figure 44 upper spectrum).

Other *CML* elements occurring commonly within sections describing spectral information are shown and shortly described in Table 3.

These elements are mostly containers for additional information on the experiment. Conditions of the experiment are to be stored in <parameterList> and <conditionList>. With the two being usable quite arbitrary, just that the first one has <parameter> children, whereas the latter might hold any *CML* elements that make sense. We think, that chemical conditions of experiments like pH, temperature and concentration are better stored within <conditionList> while <parameterList> is better used for storing e.g. machine settings.

However, as already stated, these are just proposals and not obligatory rules.

element	short description
parameterList	container for parameters
parameter	element holding parameters, like e.g. machine settings
conditionList	container for chemical conditions
metadataList	container for meta data
metadata	element holding relevant meta data
sample	container without controlled semantics, for free use by authors
scalar	primitive element for e.g. physical quantities, provided with semantics by using dictionary references
substanceList	container substances
substance	primitive element for e.g. physical quantities, provided with semantics by using dictionary references

Table 3: CML elements commonly occurring in CML encoded spectral data.

<substance> refers to a physical material. If contained in a <spectrum> it is the material that was or is studied. If there are mixtures of substances a <substanceList> with regarding <substance> children should be used. Another important and probably very intensively used element group within *CML* spectra is the <metadataList> and its <metadata> children. We defined <metadata> in comparison to <parameter> and <condition> as something the library/web community care about, while the latter is probably more used by processing software. In *Bioclipse* we use <metadata> e.g. for user defined fields to be added to the spectrum and to include all that data existing in many *JCAMP-DX* encoded spectra, that does not fit into any of the other container elements (see also Chapter 5.2.3, Figure 20, and Chapter 5.2.2). Beside these elements, it is certainly possible and very often useful to include other *CML* elements into spectral sections. So it makes sense to add connection tables describing molecules or add *CMLReact* encoded reaction sections for describing e.g. fragmentation processes as they occur in mass spectrometry.

There exists no general agreement on the definition and interpretation of a spectrum. However, software implemented for parsing spectral data does need to rely on the existence or absence of certain information. As there is no general *XML* mechanism available to address this problem, a new approach was defined using *Schematron*, *XSLT* and *XPath*. No general *CMLSpect* processor can support the huge diversity of constraints and conversions needed for spectral data, especially as many of these may not even be communally agreed. To solve this, the concept of conventions was introduced and two examples being set up for *NMRShiftDB* (an open access online database for *NMR* spectra) and *JSpecView* (an open access spectrum visualisation application). Both these conventions are defined as informal and human readable documents, and as computer readable *Schematron* files to be used for automatic validation. Within these files rules, like e.g. if the spectrum is continuous and 1 dimensional, the number of x- and y-values must identical, are defined (see Chapter 9, page 120 for an example).

To add semantic relationships to *CMLSpect* and to uniquely define certain entries the already mentioned dictionaries are used. These dictionaries are thought to develop separately of the schema and are be build by a large number of entries defining the available terms for a certain community. As an example a *JCAMP-DX* dictionary was developed during this thesis (see Figure 18). This dictionary provides an exhaustive listing of defined *JCAMP-DX* identifiers and can be used to mark entries within a *CMLSpect* file as *JCAMP-DX* originating and/or compliant. Beside this, *CML* comes with a variety of different dictionaries delivering collections of term definitions for a broad range of chemically relevant terms. Although the support of conventions and dictionaries already allows to structure the data and add meaning to the single entries, we hope that there will an ontology be developed by the chemoinformatics community to ensure the unique definition not just of the used terms, but of the hierarchical dependencies between them as well.

7 Conclusions & Outlook

Small organic molecules are of major importance in many different fields of chemistry, biology and biochemistry. They play crucial roles in metabolic networks, are needed in material sciences and are produced and consumed on a daily basis in chemical and biological laboratories. In addition, the knowledge about their interaction helps us to understand biological systems and organisms.

In the course of this project algorithms and applications, supporting scientists in their daily work on small organic molecules, were developed. A focus was put on storage, analysis, searching, exchange and encoding of molecular data under preservation of its semantics. Most often a compound cannot be described directly by e.g. determining its 3D coordinates and performing all the necessary experiments to assure its functionality. Therefore, it is common to describe a molecule by its properties and try to back reference to its functionality and structure by comparison with a set of known compounds. A big fraction of data, which is recorded in this context, is resulting from spectroscopic experiments. Consequently, the tools developed concentrate on handling structural and spectroscopic information. Fundamental functionality, like structure and spectrum visualisation and manipulation, editing of meta data and properties, import and export to a variety of spectral and structural data formats and the creation of new spectral and structural objects was implemented (see Chapters 5.1 and 5.2). This forms the basis for the development of more advanced and abstract functionality based on the provided data structures, algorithms and overall framework.

Some examples of tools extending this elementary functionality, like the assignment of structural and spectral features to each other and some analysis methods, were implemented within this thesis as well (see Chapter 5.3). Others are still under development or in the planning phase. The implementation of the assignment module in its current version allows the client based processing of *NMR* data for integration into the online *NMR* database *NMRShiftDB*. Beside this another module was implemented, enabling the integration of *BibTeX* based bibliographic data into the data set.

The general spectrum support is currently being enhanced by two ongoing projects. The first is integrating an application for the prediction of mass spectra by simulating the ionisation and the fragmentation process occurring on measuring mass spectra. The method is using a machine learning approach combined with fragmentation rules to realise the prediction. Miguel Rojas, who developed this application, is at the moment integrating it into the overall framework to control the process and visualise the results from within the *Graphical User Interface*. Therefore, an additional object for handling reactions in general was added to the system, and therewith available for other modules as well.

Another project is integrating the *SENECA* program package for *Computer Assisted Structure Elucidation* (*CASE*) of organic molecules into the framework [25]. *SENECA* uses spectroscopic data, mainly from 1D and 2D *NMR* experiments, to stochastically generate all structures meeting extracted restrictions. This process is guided to a global optimum by using a simulated annealing algorithm. The program package is at the moment adapted and integrated into the framework. Both applications are using the elementary functionality developed within the course of this project to parse spectral data, represent it within the system and to visualise their results.

Additionally, it is planned to extend the analysis functionality of the spectrum handling module. In this context, methods to perform and visualise peak integration, a mathematical method allowing to directly compare the size (in form of area under the curve) of different peaks will be added. This allows e.g. in 1D *NMR* spectroscopy to understand, how many of the observed nuclei give rise to this peak, as the size of it is proportional to this number.

A further reasonable extension of the analysis package would be a peak annotation tool kit. This tool kit would allow for the graphical annotation of a spectrum by marking regions as being of special interest and would give a user the possibility to have his/her conclusions written to the underlying object together with this marker. This would allow e.g. for labelling peaks within a *NMR* spectrum, that are resulting from the spin-spin coupling (also called j-coupling or scalar coupling) between *NMR* active nuclei. This phenomenon arises from the interaction of different spin states, and results in a splitting of signals into recognisable patterns. The storage of this annotation is completely supported by the *CMLSpect* format used for spectrum handling.

Beside this, the support of data resulting from the combination of chromatographic experiments and mass spectrometry would be very valuable, as this procedure is intensely used in *Metabolomics* and systems biology for the identification of unknown substances. This would, beside the implementation of an adequate resource schema, need an adaptation of the current spectrum visualisation to allow for the selection of single signals within a chromatogram, which should trigger the regarding mass spectrum to be opened as well.

The developed tools allow for the structured deposition and management of large amounts of molecular data within the file system as well as in associated databases. The actual connection of different *SQL* based *Relational Database Management Systems* (*RDBMS*) was realized by using the *Hibernate* object-relational mapping system. A complex schema was implemented to realise the database integration without loosing the component independence of the system. Together these two factors allow for flexible use of a variety of database systems within the *Bioclipse* application and liberate plug-in developers from the burden of solving the object-relational mismatch. Persistence of molecules, spectra and combined objects (enhanced with assignments and annotations) into the *Hypersonic SQL Database* was implemented and example queries were added to the system (see Chapter 5.4).

The implemented framework for connecting existing modules to all major *RDBMS's* in use provides the basis for integrating advanced database functionality. It is planned to develop a system to retrieve data from connected databases by using wizards for allowing the user to construct database queries from a set of predefined terms. This could be imagined to be realised in the same way, as it is solved for most online databases. These systems give users the possibility to select from an ordered display of all single terms stored within the database and let them combine these by using general logical operations supported by the databases. An additional possibility would be to pre-compile actions for complex queries, that are expected to be frequently used and add them to the general action framework.

Beside this, the existing integration of web services for accessing remote data repositories should be extended and improved. At the moment, there are modules under development, that allow for accessing *NMRShiftDB* via webservices for submitting data sets, performing a *NMR* spectrum prediction for available structures and realising an automatic assignment of structural data to *NMR* spectral data. These modules will be expanded to allow for structured searching for as well molecules as spectra within the database. It is planned to release a compiled collection of modules forming a *NMRShiftDB* client, as an alternative to the web front end to be used for submission and review of local data.

Additionally, there already exists a component providing access to a variety of online databases hosted by the *EBI* (see Chapter 4). At the moment, it is just possible to retrieve data by its identifier. This is planned to be improved in a way allowing for the integration of these remote databases to the general search framework drafted some sentences ago. Apart from this, an access to further databases, like *PubChem*, *Zinc* and *PDB* is planned to be integrated into the framework. These could then e.g. be used for performing similarity searches or for the retrieval of additional information for an object under analysis.

There is a growing need to store data in communally agreed data formats. Many important information about molecules on a structural level are measured by using spectroscopic methods. As there was no open, standardised and structured data format available allowing for lossless storage of this type of data under preservation of semantics, the available *Chemical Markup Language* (*CML*) was expanded by the definition of a vocabulary for spectral data - *CMLSpect*. The design of this format enables for encoding primary spectroscopic data as well as processed spectra in form of peak lists. Additionally, it is possible to add various meta information, about the compound measured, the experiment itself, the apparatus used and other general meta data, to the spectral data. Furthermore, it provides places were annotations can be added to data elements without defining a stringent vocabulary for doing this. This is necessary, as spectroscopists and chemists use many different ontologies, which may be fundamentally irreconcilable. To ensure unique definition of the used terms a set of dictionaries is provided with *CML* and their use is enabled and propagated within the definition. Beside the existing *CML* dictionaries an additional one was developed, that allows to mark data within a *CMLSpect* file as *JCAMP-DX* compliant or originating and unambiguously defines its meaning. Together with the general possibility to combine different *CML* elements, and the provided functionality to assign elements to each other by references, a format was developed, which allows the flexible encoding of most spectral data in use (see Chapter 6).

Most of the functionality developed was integrated into the *Bioclipse* framework for chemo- and bioinformatics. This framework is a *Rich Client* implementation based on the *Eclipse Rich Client Platform* and therefore inherits the advantages of this system. Thus, the functionality is presented to the user embedded into a high performance, high quality and easy to use *Graphical User Interface*. On a programmatic level the application structure based on an overall plug-in architecture allows for easy expandability by ensuring high data consistency.

The *Bioclipse* application in its current status is already integrating much of the software needed within the information work flow in life sciences presented in Figure 39 and Figure 38. A complete package of applications integrated into the framework supporting all steps, from the initial data generation up to the publication of results, is easily imaginable. Several of the currently developed components will be of major help within this context. These just have to be extended by e.g. an easy to use *CML* editor providing an appealing interface for creating *CML* encoded data files. This probably should look more like a word processor than like a typical *XML* editor. If this editor would provide functionality expected from a general word processor as well, it would be possible to use the framework to create *XML* based documents suited for the publication process.

Another, probably a lot easier to implement, option would be to integrate an existing word processor or office suite into the system. The *OpenOffice* project would be the appropriate choice, as it is released under *LGPL* and provides a *Java* interface. By connecting office based documents with molecular information, both stored within the system, it would be possible to generate semantically rich documents that could be used for publishing.

A further option would be to extend the system in such a way, that it could be used as a *Electronic Lab Notebook* (*ELN*). This would need the development of modules allowing for the structured documentation of experiments and procedures in laboratories. By adding a user management and modules for the definition of standard operation procedures even a whole *Laboratory Information Management System* (*LIMS*) would be conceivable.

There is no other freely available software system known to the author, that provides comparable functionality in such an integrated way as the described application. In both bioinformatics and chemoinformatics there exist several open source projects, which provide functionality for special sub fields (e.g. *OpenBabel* [111], *KNIME* [112], *Taverna* [113], *TOUCAN* [114]). Some others are integrating more diverse functionality (e.g. *BioJava* [115], *BioPython* [116]) or provide loosely coupled frameworks for integrating existing software modules (e.g. *EMBOSS* [117], *ISYS* [118]). But none of these is integrating bio- and chemoinformatics functionality into one workbench with a high quality, intuitive user interface and a powerful plug-in architecture for easy expandability.

There are some commercial packages on the market, which provide an exhaustive set of functionality embedded into a highly integrated framework (e.g. *Discovery Studio* from *Accelrys*, *Moe* from the *Chemical Computing Group* and the program packages provided by *ACDLabs*), but these are not based on open standards, are not open source and normally are quite expensive to purchase. So they do neither allow for easy exchange of generated or analysed data nor for simple extension of the framework by additional modules. Therein lies one of the major advantages of the components developed within this project, they are freely available and they can, from their very first implementation on, be scrutinised by interested parties. Additionally, because of the well defined extension mechanisms, it is facile to integrate new functionality via new modules into the overall system as well as to compile new module subsets meeting the actual demands. The usage of open, standardised and structured data formats facilitates the exchange of utilised or generated data under preservation of semantics.

Although the *Bioclipse* framework is a relatively young project, it was awarded third price and the audience award at the *JAX Innovation Award 2006*, which is “intended to honour and recognise the most remarkable and outstanding contributions in the world of *Java* and *Eclipse*” [119].

8 References

1. Rahman, S. A. & Schomburg, D.: **Observing local and global properties of metabolic pathways: 'load points' and 'choke points' in the metabolic networks**. *Bioinformatics* 2006, **22**:1767-1774.
2. Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., et al.: **The Protein Data Bank**. *Nucleic Acids Research* 2000, **28**:235-242.
3. Kanehisa, M., Goto, S., Hattori, M., Aoki-Kinoshita, K.F., Itoh, M., et al.: **From genomics to chemical genomics: new developments in KEGG.**. *Nucleic Acids Research* 2006, **34**:D354-357.
4. Bairoch, A., Apweiler, R., Wu, C.H., Barker, W.C., Boeckmann, B., et al.: **The Universal Protein Resource (UniProt).**. *Nucleic Acids Research* 2005, **33**:D154-D159.
5. **RSC Publishing Pioneers Next Generation Of Enriched Articles** [http://www.rsc.org/Publishing/Journals/News/launch.asp]. accessed February 2007.
6. Butcher, E. C. & Berg, Ellen L. & Kunkel, E.J.: **Systems biology in drug discovery**. *nature biotechnology* 2004, **22**:1253-1259.
7. Klipp, E., Herwig, R., Kowald, A., Wierling, C. & Lehrach, H.: **Systems Biology in Practice**. *Wiley-VCH Verlag GmbH & Co. KGaA, Weinheim*; 2005.
8. Kitano, H.: **Systems Biology: a brief overview**. *Science* 2002, **295**:1662-1664.
9. Lander, E. S., Linton, L. M., Birren, B., Nusbaum, C., Zody, M. C. et al.: **Initial sequencing and analysis of the human genome**. *Nature* 2001, **409**:860-921.
10. Aderem, A: **Systems Biology: its practice and challenges**. *Cell* 2005, **121**:511-513.
11. Kitano, H.: **Computational systems biology**. *Nature* 2002, **420**:206-210.
12. Hucka, M., Finney, A., Sauro, H.M., Bolouri, H., Doyle, et al.: **The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models**. *Bioinformatics* 2003, **19**:524--531.
13. Ideker, T., Galitski, T. and Hood L.: **A New Approach To Decoding Life: Systems Biology**. *Annual Review of Genomics and Human Genetics* 2001, **2**:343-372.
14. Fiehn, O.: **Metabolomics-the link between genotypes and phenotypes**. *Plant Molecular Biology* 2002, **48**:155-171.
15. Mendes, P.: **Emerging bioinformatics for the metabolome**. *Briefings in Bioinformatics* 2002, **3**:134-145.
16. Nobeli, I. & Thornton, J.M.: **A bioinformatician's view of the metabolome**. *BioEssays* 2006, **28**:534-545.
17. Brindle, J. T., Antti, H., Holmes, E., Tranter, G., Nicholson, J. K. et al.: **Rapid and noninvasive diagnosis of the presence and severity of coronary heart disease using 1H-NMR-based metabonomics**. *Nature Medicine* 2002, **8**:1439-1444.
18. Steinbeck, C.: **The automation of natural product structure elucidation**. *Current Opinion in Drug Discovery & Development* 2001, **4**:338-342.
19. Steinbeck, C.: **Computer-Assisted Structure Elucidation. Handbook on Chemoinformatics**. *Wiley-VCH Weinheim*; 2003.

20.Munk, M.: **Computer-Based Structure Determination: Then and Now**. *Journal of Chemical Informatics and Computer Sciences* 1998, **38**:997-1009.

21.Steinbeck, C.: **Recent developments in automated structure elucidation of natural products**. *Natural Product Reports* 2004, **21**:512-518.

22.Neudert, R. & Penk, M.: **Enhanced Structure Elucidation**. *Journal of Chemical Informatics and Computer Sciences* 1996, **36**:244-248.

23.Grey, N.: **Computer Assisted Structure Elucidation**. *John Wiley & Sons*; 1986.

24.Funatsu, K., Miyabayashi, N. & Sasaki, S.I.J.: **Further Development of Structure Generation in the Automated Structure Elucidation System CHEMICS**. *Journal of Chemical Informatics and Computer Sciences* 1988, **22**:18-28.

25.Steinbeck C: **SENECA: A platform-independent, distributed, and parallel system for computer-assisted structure elucidation in organic chemistry**. *Journal of Chemical Informatics and Computer Sciences* 2001, **41**:1500-1507.

26.Gasteiger, J., Hanebeck, W. & Schulz, K.P.: **Prediction of Mass Spectra from Structural Information**. *Journal of Chemical Informatics and Computer Sciences* 1992, **32**:264-271.

27.Affolter, C. & Clerc, J.T.: **Prediction of Infrared Spectra from Chemical Structures of Organic Compounds using Neural Networks**. *Chemometrics and Intelligent Laboratory Systems* 1993, **21**:151-157.

28.Hesse, M., Meier, H. & Zeeh, B.: **Spektroskopische Methoden in der organischen Chemie**. *Thieme, Stuttgart*; 2005.

29.**GNU Lesser General Public License** [http://www.gnu.org/licenses/lgpl.html]. accessed January 2007.

30.**The Analytical Data Interchange Standard Webpage** [http://andi.sourceforge.net/]. accessed February 2007.

31.**Thermo Galactic Spc File Format Webpage** [http://www.thermo.com/com/cda/resources/resources_detail/1,,112125,00.html]. accessed February 2007.

32.**The Open Source Initiative** [http://opensource.org/]. accessed January 2007.

33.Raymond, E.: **The Cathedral & the Bazaar**. *O'Reilly Media*; 2001.

34.**The Budapest Open Access Initiative** [http://www.soros.org/openaccess/]. accessed January 2007.

35.Guha, R., Howard, M.T., Hutchison, G.R., Murray-Rust, P., Rzepa, H., et al.: **The Blue Obelisks Interoperability in Chemical Informatics**. *Journal of Chemical Information and Modelling* 2006, **46**:991-998.

36.**The OpenSciene Project** [http://www.openscience.org/]. accessed January 2007.

37.Schussel, G.: **Client/Server: Past, Present and Future**. [http://www.dciexpo.com/geos/dbsejava.htm]. accessed January 2007.

38.**The Eclipse Webpage** [http://www.eclipse.org]. accessed December 2006.

39.Daum, B.: **Rich-Client-Entwicklung mit Eclipse 3.1**. *dpunkt.verlag*; 2005.

40.Klinkert, T. & Bertschler, M.: **Die Eclipse Rich-Client-Platform (RCP)**. [http://www.saboracaferestaurant.com/mb/document.pdf]. 2006.

41.**Open Service Gateway Initiative** [http://www.osgi.org]. accessed December 2006.

42.**The Eclipse Wiki** [http://wiki.eclipse.org/index.php/]. accessed December 2006.

43.Gunther, J.: **Introduction to Eclipse's Rich Client Platforms**. *JavaOne Conference*; 2005.

44.des Rivieres, J. & Beaton, W.: **Eclipse platform technical overview**. [http://www.eclipse.org/articles/Whitepaper-Platform-3.1/]. accessed December 2006.

45.Szyperski, C.: **Component software: beyond object-oriented programming**. *Addison-Wesley Professional, Boston*; 2002.

46.Arthorne, J. & Laffra, C.: **Official Eclipse 3.0 FAQs**. *Addison Wesley Professional*; 2004.

47.**The Eclipse 3.2 Help System** [http://help.eclipse.org/help31]. accessed December 2006.

48.**The SWT Webpage** [http://www.eclipse.org/swt/]. accessed December 2006.

49.Marinilli, M.: **Swing and SWT: a tale of two Java GUI libraries**. [http://www.developer.com/]. accessed December 2006.

50.Spjuth, O., Helmus, T., Willighagen, E.L., Kuhn, S., Eklund, M., et al.: **Bioclipse: An open source workbench for chemo- and bioinformatics**. *BMC Bioinformatics* 2007, **8**:59.

51.**The BioJava Webpage** [http://biojava.org]. accessed January 2007.

52.**The Jmol Webpage** [www.jmol.org]. accessed January 2007.

53.Neerincx, P. B. & Leunissen, J. A.: **Evolution of web services in bioinformatics**. *Briefings in Bioinformatics* 2005, **6**:178-188.

54.Curcin, V., Ghanem, M. & Guo, Y.: **Web services in the life sciences**. *Drug Discovery Today* 2005, **10**:865-871.

55.Pillai, S., Silventoinen, V., Kallio, K., Senger, M., Sobhany, S. et al.: **SOAP-based services provided by the european bioinformatics institute**. *Nucleic Acids Research* 2005, **33**:w25-w28.

56.Murray-Rust, P., Rzepa, H., Williamson, MJ & Willighagen, E.L.: **Chemical markup, XML, and the World Wide Web. 5. Applications of chemical metadata in RSS aggregators**. *Journal of Chemical Informatics and Computer Sciences* 2004, **44**:462-469.

57.**Rich Site Summary (RSS 0.91) Definition.** [http://backend.userland.com/rss091]. accessed January 2007.

58.**Rich Site Summary (RSS 0.92) Definition.** [http://backend.userland.com/rss092]. accessed January 2007.

59.**The Rdf Site Summary (RSS 1.0) Definition.** [http://web.resource.org/rss/1.0/spec]. accessed January 2007.

60.**Really Simple Syndication (RSS 2.0) Definition.** [http://www.rssboard.org/rss-specification]. accessed January 2007.

61.Steinbeck, C. & Kuhn, S.: **NMRShiftDB -- compound identification and structure elucidation support through a free community-built web database**. *Phytochemistry* 2004, **65**:2711-2717.

62.Steinbeck, C., Krause, S., Kuhn, S.: **NMRShiftDB - constructing a free chemical**

information system with open-source components. *Journal of Chemical Informatics and Computer Sciences* 2003, **43**:1733-1739.

63.Steinbeck, C., Han, Y., Kuhn, S., Horlacher, O. & Luttmann, E. et al.: **The Chemistry Development Kit (CDK): an open-source Java library for Chemo- and Bioinformatics**. *Journal of Chemical Informatics and Computer Sciences* 2003, **43**:493-500.

64.Krause, S., Willighagen, E. & Steinbeck, C.: **JChemPaint - Using the Collaborative Forces of the Internet to Develop a Free Editor for 2D Chemical Structures**. *Molecules* 2000, **5**:93-98.

65.Murray-Rust, P. & Rzepa, H.S.: **Chemical Markup, XML, and the Worldwide Web. 1.Basic Principles**. *Journal of Chemical Informatics and Computer Sciences* 1999, **39**:928-942.

66.McDonald, R.S. & Wilks, P.A. Jr.: **JCAMP-DX: A Standard Form for Exchange of Infrared Spectra in Computer Readable Form**. *Applied Spectroscopy* 1988, **42**:151-162.

67.Lampen, P., Hillig, H., Davies, A.N. & Linscheid, M.: **JCAMP-DX for Mass Spectrometry**. *Applied Spectroscopy* 1994, **48**:1545-1552.

68.Lampen, P., Lambert, J., Lancashire, R.J., McDonald, R.S., McIntyre, P.S., et al.: **An Extension to the JCAMP-DX Standard File Format, JCAMP-DX V.5.01**. *Pure and Applied Chemistry* 1999, **71**:1549-1556.

69.Duckworth, J.: **An XML-Based File Format for Archival Storage of Analytical Instrument Data**. [http://www.gaml.org/Documentation/XML Analytical Archive Format]. 2001.

70.**The JFreeChart Library** [http://www.jfree.org/jfreechart/]. accessed January 2007.

71.Murray-Rust, P. & Rzepa, H.S.: **Chemical markup, XML and the World-Wide Web. 2. Information objects and the CMLDOM**. *Journal of Chemical Informatics and Computer Sciences* 2001, **41**:1113-1123.

72.Kemper, A. & Eickler, A.: **Datenbanksysteme. Eine Einführung**. *Oldenbourg Verlag*; 2006.

73.Beeger, R.F., Haase, A., Roock, S. & Sanitz, S.: **Hibernate. Persistenz in Java-Systemen mit Hibernate 3**. *dpunkt.verlag*; 2006.

74.Fussell, M.: **Foundations of Object-Relational Mapping**. [http://www.chimu.com/publications/objectRelational/]. accessed January 2007.

75.**The Hibernate Webpage** [ww.hibernate.org]. accessed January 2007.

76.Elliot, J.: **Hibernate: A Developer's Notebook**. *O'Reilly Verlag*; 2004.

77.Murray-Rust, P., Mitchell, J.B. & Rzepa, H.S.: **Communication and re-use of chemical information in bioscience**. *BMC Bioinformatics* 2005, **6**:1-15.

78.Berners-Lee, T. and Hendler, J.: **Publishing on the semantic web**. *Nature* 2001, **410**:1023-1024.

79.**The Semantic Web Activity** [http://www.w3.org/2001/sw/]. accessed February 2007.

80.W3Consortium: **The Extensible Markup Language (xml)**. [http://www.w3.org/XML/]. accessed January 2007.

81.Chaudhri, A.B., Rashid, A. & Zicari R.: **XML Data Management: Native XML and XML-Enabled Database Systems**. *Addison Wesley Professional*; 2003.

82.Harold, E.R. & Means, W.S.: **XML in a Nutshell, 3rd Edition**. *O'Reilly*; 2004.

83.Fallside, D.C. & Walmsley, P.: **XML schema part 0: primer second edition**. [http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/]. accessed January 2007.

84.Thompson, H.S., Beech, D., Mendelsohn, N., Maloney M.: **XML schema part 1: structures second edition**. [http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/]. accessed January 2007.

85.Biron, P.V. and Malhotra, A.: **XML schema part 2: datatypes second edition**. [http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/]. accessed January 2007.

86.Bray, T., Hollander, D., Layman, A. and Tobin, R.: **Namespaces in XML 1.0**. [http://www.w3.org/TR/REC-xml-names/]. accessed January 2007.

87.DeRose, S., Maler, E. and Orchard, D.: **XML Linking Language (XLink) Version 1.0**. [http://www.w3.org/TR/xlink/]. accessed January 2007.

88.DeRose, S., Maler, E. and Interwoven, R.D. Jr.: **XML Pointer Language (XPointer) Version 1.0**. [http://www.w3.org/TR/WD-xptr]. accessed January 2007.

89.Clark, J. & DeRose, S.: **XML Path Language (XPath) Version 1.0**. [http://www.w3.org/TR/xpath]. accessed January 2007.

90.**The Extensible Style Sheet Language Family** [http://www.w3.org/Style/XSL/]. accessed January 2007.

91.Boag, S., Chamberlin, S., Fernández, M.F., Florescu, D., Robie, J., et al.: **XQuery 1.0: An XML Query Language**. [http://www.w3.org/TR/xquery/]. accessed January 2007.

92.Barillot, E. & Achard, F.: **XML: a lingua franca for science?**. *Trends in Biotechnology* 2000, **18**:331-333.

93.**Bioinformatic Sequence Markup Language** [http://www.bsml.org/]. accessed February 2007.

94.Fenyo, D.: **The Biopolymer Markup Language**. *Bioinformatics* 1999, **15**:339-340.

95.Finney, A. & Hucka, M.: **Systems Biology Markup Language: level 2 and beyond**. *Biochemical Society transactions* 2003, **31**:1472-1473.

96.**The PDBML Resources Webpage** [http://pdbml.rcsb.org/]. accessed January 2007.

97.**The Analytical Markup Language Webpage** [http://animl.sourceforge.net/]. accessed January 2007.

98.Murray-Rust, P. & Rzepa, H.S.: **Chemical markup, XML, and the World Wide Web. 4. CML schema**. *Journal of Chemical Informatics and Computer Sciences* 2003, **43**:757-772.

99.Achard, F., Vaysseix, G. & Barillot, E.: **XML, bioinformatics and data integration**. *Bioinformatics* 2001, **17**:115-125.

100.Cerami, E.: **XML for bioinformatics**. *Springer, Berlin*; 2005.

101.**The CML FAQ** [http://cml.sourceforge.net/historical/faq.html]. accessed January 2007.

102.Gkoutos, G. V., Murray-Rust, P. & Rzepa, H S & Wright, M: **Chemical markup, XML and the World-Wide Web. 3. Toward a signed semantic chemical web of**

trust. *Journal of Chemical Informatics and Computer Sciences* 2001, **41**:1124-1130.

103.Bartel, M., Boyer, J., Fox, B., LaMacchia, B. & Simon, E.: **XML-Signature Syntax and Processing**. [http://www.w3.org/TR/xmldsig-core/]. accessed January 2007.

104.Holliday, G. L., Murray-Rust, P. & Rzepa, H.S.: **Chemical markup, XML, and the world wide web. 6. CMLReact, an XML vocabulary for chemical reactions**. *Journal of Chemical Informatics and Computer Sciences* 2006, **46**:145-157.

105.Murray-Rust, P. & Rzepa, H.S.: **STMML. A Markup Language for Scientific, Technical and Medical Publishing**. 2002, **1**:1-65.

106.Murray-Rust, P. & Rzepa, H.S.: **Scientific publications in XML - towards a global knowledge base**. 2002, **1**:84-98.

107.V Gkoutos, G., Murray-Rust P. Rzepa H.S. Viravaidyaa C. & Wright, M.: **The Application of XML Languages for Integrating Molecular Resources**. *Internet Journal of Chemistry* 2001, **article 13**:1-15.

108.**The XML Object Model (XOM) Webpage** [http://www.xom.nu/]. accessed January 2007.

109.Adams, S.E., Goodman, J.M., Kidd, R.J. McNaught, A.D., Murray-Rust, P. , et al.: **Experimental data checker: better information for organic chemists**. *Organic & Biomolecular Chemistry* 2004, **2**:3067-3070.

110.Townsend, J.A., Adams, S.E., Waudby, C.A., de Souza, V.K., Goodmann, J.M. et al.: **Chemical documents: machine undertsanding and automated information extraction**. *Organic & Biomolecular Chemistry* 2004, **2**:3294-3300.

111.**OpenBabel** [http:/en.wikipedia.org/wiki/OpenBabel]. accessed March 2007.

112.**KNIME** [http://www.knime.org/]. accessed March 2007.

113.Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., et al.: **Taverna: a tool for the composition and enactment of bioinformatics workflows**. *Bioinformatics* 2004, **20**:3045-3054.

114.Aerts, S., Van Loo, P., Thijs, G., Mayer, H., de Martin, R., et al.: **TOUCAN 2: the all-inclusive open source workbench for regulatory sequence analysis**. *Nucleic Acids Res* 2005, **33**:W393-6.

115.**BioJava** [http://biojava.org/]. accessed March 2007.

116.**BioPython** [http://biopython.org/]. accessed March 2007.

117.Rice, P., Longden, I. & Bleasby, A.: **EMBOSS: the European Molecular Biology Open Software Suite**. *Trends Genet* 2000, **16**:276-277.

118.Siepel, A., Farmer, A., Tolopko, A., Zhuang, M., Mendes, P., et al.: **ISYS: a decentralized, component-based approach to the integration of heterogeneous bioinformatics resources**. *Bioinformatics* 2001, **17**:83-94.

119.**The JAX Innovation Award 2006** [http://jax-award.de/jax_award06/index_en.php]. accessed March 2007.

9 Appendix

CML Examples

```
<cml xmlns="http://www.xml-cml.org/schema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:siUnits="http://www.xml-cml.org/units/siUnits"
  xmlns:units="http://www.xml-cml.org/units/units"
  xmlns:jspecview="http://jspecview.sf.net/convention.html"
  xmlns:cmlDict="http://www.xml-cml.org/dict/cmlDict"
  xmlns:cml="http://www.xml-cml.org/dict/cml"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:jcamp="http://www.xml-cml.org/dict/jcampDict"
  xsi:schemaLocation="http://www.xml-cml.org/dict/jcampDict
  dict/jcampDict.xml http://www.xml-cml.org/schema schema.xsd http://www.xml-
  cml.org/dict/cml dict/cmlDict.xml http://www.xml-cml.org/dict/cmlDict
  dict/simpleCmlDict.xml http://www.xml-cml.org/units/units dict/unitsDict.xml
  http://www.xml-cml.org/units/siUnits dict/siUnitsDict.xml">
  <spectrum id="UV_VIS" title="Holmium Oxide Wavelength Standard"
    convention="JSpecView" type="UV/VIS">
    <metadataList>
      <metadata name="jcamp:origin" content="Lambda 900" />
      <metadata name="jcamp:owner" content="NIST-Gaithersburg" />
    </metadataList>
    <parameterList>
      <parameter dictRef="jcamp:SpectrometerDataSystem"
        title="SpectrometerDataSystem" value="LAMBDA" />
      <parameter dictRef="jcamp:resolution" title="resolution">
        <scalar units="units:nm">2.0 NM</scalar>
      </parameter>
    </parameterList>
    <sample>
      <molecule>
        <formula inline="Ho2O3" />
        <name convention="cml:casregno">12055-62-8</name>
      </molecule>
    </sample>
    <spectrumData>
      <xaxis>
        <array units="units:nm" start="200.0" end="700.0" size="501"
          dataType="xsd:double">
          200 201
           ...
           699 700
        </array>
      </xaxis>
      <yaxis multiplierToData="1.000">
        <array units="cml:absorbance" size="501"
          dataType="xsd:double">
          0.46341657 0.44507496
           ...
           0.02941928 0.0297209
        </array>
      </yaxis>
    </spectrumData>
  </spectrum>
</cml>
```

Figure 45: *CMLSpect* encoded *UV/Vis* spectrum exported from *JSpecView* as defined in the convention as well. The data blocks showing the x and y data points were shortened and do just show the first and the last two entries. It contains meta data, parameter data and a sample beside the continuous spectrum itself, defined within the <spectrumData> block. Furthermore, several dictionaries are used to unambiguously define most of the utilised terms. Via the <sample> element the measured sample is described by its formula and a *CAS* registry number.

```
<spectrum id="but2" title="2-Butanol" convention="JSpecView"
  type="infrared" state="gas" xmlns="http://www.xml-cml.org/schema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:siUnits="http://www.xml-cml.org/units/siUnits"
  xmlns:units="http://www.xml-cml.org/units/units"
  xmlns:jspecview="http://jspecview.sf.net/convention.html"
  xmlns:cmlDict="http://www.xml-cml.org/dict/cmlDict"
  xmlns:cml="http://www.xml-cml.org/dict/cml"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xsi:schemaLocation="http://www.xml-cml.org/units/siUnits dict/siUnitsDict.xml
  http://www.xml-cml.org/units/units dict/unitsDict.xml http://www.xml-
  cml.org/dict/cmlDict dict/simpleCmlDict.xml http://www.xml-cml.org/dict/cml
  dict/cmlDict.xml http://www.xml-cml.org/schema schema.xsd">
  <metadataList>
    <metadata name="dc:origin"
      content="Sadtler Research Labs Under US-EPA Contract" />
    <metadata name="dc:owner" content="NIST Standard Reference Data Program" />
    <metadata name="dc:identifier" content="No.424 (EPA Vapor Library" />
    <metadata name="dc:source"content="I am guessing this means vapor phase" />
  </metadataList>
  <sample>
    <molecule>
      <formula concise="C 4 H 10 O 1" />
      <name convention="cml:casregno">78-92-2</name>
    </molecule>
  </sample>
  <conditionList>
    <scalar dictRef="units:bar">1.2345</scalar>
    <scalar dictRef="cmlDict:press" units="siUnits:pascal">12345</scalar>
  </conditionList>
  <spectrumData>
    <xaxis>
      <array units="units:cm-1" size="224" dataType="xsd:double">
        450 454
       ...
       1338 1342
      </array>
    </xaxis>
    <yaxis multiplierToData="0.000109021">
      <array units="cml:absorbance" size="224" dataType="xsd:double">
        331 179
       ...
       0 0
      </array>
    </yaxis>
  </spectrumData>
  <peakList>
    <peakGroup id="pg1" xMax="3040" xMin="2800">
      <peak id="ch1" title="CH-stretch-1" peakMultiplicity="singlet"
        peakShape="sharp" xUnits="units:cm-1" xValue="2974"
        yUnits="cml:absorbance" yValue="1.0921" />
      <peak id="ch2" title="CH-stretch-2" peakShape="shoulder"
        xUnits="units:cm-1" xValue="2938" yUnits="cml:absorbance"
        yValue="0.653" />
      <peak id="ch3" title="CH-stretch-3" xUnits="units:cm-1"
        xValue="2890" yUnits="cmlp:absorbance" yValue="0.470" />
    </peakGroup>
    <peak id="oh1" title="CH-stretch???" peakShape="broad"
      xUnits="units:cm-1" xValue="3657" yUnits="cml:absorbance"
      yValue="0.1092" />
  </peakList>
</spectrum>
```

Figure 46: From *JSpecView* exported *IR* spectrum. Via the <sample> entry the measured sample is described by its formula and a *CAS* registry number. The spectrum contains as well continuous data (for displaying purposes shortened) as extracted peaks. Three of its peaks are grouped by using the <peakGroup> element implying, that these peaks have a chemical relationship. Additionally, all peaks are further described by using the peakShape attribute. Again, many of the used terms defined using dictionary references.

```
<cml xmlns="http://www.xml-cml.org/schema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:siUnits="http://www.xml-cml.org/units/siUnits"
  xmlns:units="http://www.xml-cml.org/units/units"
  xmlns:jspecview="http://jspecview.sf.net/convention.html"
  xmlns:jcamp="http://www.jcamp.org/dict"
  xmlns:cml="http://www.xml-cml.org/dict/cmlDict"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xsi:schemaLocation="http://www.xml-cml.org/dict/cmlDict dict/simpleCmlDict.xml
  http://www.xml-cml.org/dict/jcampDict dict/jcampDict.xml http://www.xml-
  cml.org/units/units dict/unitsDict.xml http://www.xml-cml.org/schema schema.xsd">
  <spectrum id="MS_4-vinylben" title="4-vinylbenzyl chloride"
    convention="JSpecView" type="massSpectrum">
    <metadataList>
      <metadata name="dc:origin"
        content="PSLC - Univ of Wisconsin-Stevens Point" />
      <metadata name="dc:owner" content="Robert Badger" />
    </metadataList>
    <parameterList>
      <parameter dictRef="jcamp:SpectrometerDataSystem"
        title="SpectrometerDataSystem" value="unknown" />
    </parameterList>
    <sample>
      <molecule>
        <formula inline="C9H9Cl" />
        <name convention="cml:casregno">1592-20-7</name>
      </molecule>
    </sample>
    <peakList>
      <peak id="a1" xUnits="units:moverz" xValue="26.05"
        yUnits="cml:relabundance" yValue="0.86">
      </peak>
      <peak id="a2" xUnits="units:moverz" xValue="27.05"
        yUnits="cml:relabundance" yValue="2.34">
      </peak>
        ...
      <peak id="a90" xUnits="units:moverz" xValue="155"
        yUnits="cml:relabundance" yValue="2.89">
      </peak>
      <peak id="a91" xUnits="units:moverz" xValue="156"
        yUnits="cml:relabundance" yValue="0.23">
      </peak>
    </peakList>
  </spectrum>
</cml>
```

Figure 47: *CMLSpect* example for a mass spectrum. This spectrum is just holding peak information. The original spectrum contains 92 peaks, of which just 4 are shown here for demonstration. Again, the sample is described by its formula and a *CAS* registry number and the used terms are further defined via dictionary references.

NMRShiftDB Convention Schema

```
<schema xml:lang="en" version="0.2"
  xmlns="http://purl.oclc.org/dsdl/schematron">
  <!-- Copyright (c) 2006 Stefan Kuhn, Egon Willighagen -->
  <!-- Version 0.2 is the specification distributed along with the CMLSpec article.-->

  <title>CML Convention for NMRShiftDB.org</title>
  <ns prefix='cml' uri='http://www.xml-cml.org/schema'/>
  <pattern name="Convention specification">
    <rule id="conv1" context="/*">
      <assert test="@convention and @convention='nmrshiftdb-convention'">The root
       element must specify @convention="nmrshiftdb-convention".</assert>
    </rule>
  </pattern>
  <!-- hierarchy of elements -->
  <pattern name="Element Hierarchy">
    <rule id="hier1" context="/*">
      <assert test="name()='cml' or name()='molecule' or name()='spectrum'">The root
       is not <cml>, <molecule> or <spectrum>.</assert>
    </rule>
    <rule id="hier2" context="cml:cml">
      <assert test="count(cml:molecule) < 2">Only one <molecule> element may be
       present.</assert>
    </rule>
    <rule context="cml:molecule">
      <assert id="hier3" test="cml:atomArray">The <molecule> element must contain a
       <atomArray> element.</assert>
      <assert id="hier4" test="cml:bondArray">The <molecule> element must contain a
       <bondArray> element.</assert>
    </rule>
    <rule context="cml:spectrum">
      <assert id="hier5" test="cml:conditionList">The <spectrum> element must contain
       a <conditionList> element.</assert>
      <assert id="hier6" test="cml:metadataList">The <spectrum> element must contain a
       <metadataList> element.</assert>
      <assert id="hier7" test="cml:substanceList">The <spectrum> element must contain
       a <substanceList> element.</assert>
      <assert id="hier8" test="cml:peakList">The <spectrum> element must contain a
       <peakList> element.</assert>
    </rule>
  </pattern>
  <!-- element attributes -->
  <pattern name="Required attributes">
    <rule id="attr1" context="cml:atom">
      <assert test="@id">The <atom> element must have an @id attribute.</assert>
    </rule>
    <rule context="cml:spectrum">
      <assert id="attr2" test="@moleculeRef">The <spectrum> element must have an
       @moleculeRef attribute.</assert>
      <assert id="attr3" test="@type">The <spectrum> element must have an @type
       attribute.</assert>
    </rule>
    <rule context="cml:peak">
      <assert id="attr4" test="@xValue">For each <peak> the @xValue must be
       given.</assert>
      <assert id="attr5" test="@xUnits">For each <peak> the @xUnits must be
       given.</assert>
      <report id="attr6" test="@yValue and not(@yUnits)">If @yValue is given for a <
       peak>, then @yUnits must be given too.</report>
      <assert id="attr7" test="@atomRefs">For each <peak> the @atomRefs must be
       given.</assert>
    </rule>
  </pattern>
...
</schema>
```

Figure 48: Sample section of the *NMRShiftDB* convention file. By this schematron schema certain rules are defined for *CML* files to be *NMRShiftDB* conform. It is e.g. defined, that the file is only allowed to carry one molecule, that has to be formed by an atomArray and a bondArray.

Metadata Mapping File Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<dictionaryMapping prefix="jcampdx"
  xmlns:jcampdx="http://www.xml-cml.org/dict/jcampDXDict"label="JCAMP-DX Metadata
  Entries" id="JCAMP-DX" dictLocation="/dict10/simple/">
  <section name="conditionList" label="Condition List">
    <entry id="TEMPERATURE" allowedForSpecTypes=""
      label="Temperature ">
    </entry>
    <entry id="dotIONIZATIONMODE" allowedForSpectrumTypes="MS"
      label="Ionization Mode ">
    </entry>
    ...
  </section>
  </section>
  <section name="substanceList" label="Substance List">
    <entry id="dotSOLVENTNAME" allowedForSpecTypes="NMR,MS,IR"
      label="Solvent Name ">
      <valueList>
        <value>H2O</value>
        <value>PyridineD5</value>
        <value>Benzene</value>
        <value>THF</value>
      </valueList>
    </entry>
    ...
  </section>
  <section name="metadataList" label="Metadata List">
    <entry id="XUNITS" allowedForSpectrumTypes="NMR,MS,IR"
      label="XUnits ">
      <valueList>
        <value>M/Z</value>
        <value>ppm</value>
        <value>nm</value>
        <value>HZ</value>
        <value>SECONDS</value>
      </valueList>
    </entry>
  </section>
</dictionaryMapping>
```

Figure 49: Depiction showing sections of the *JCAMP-DX* meta data mapping file used to build the meta data editor. At the moment there are three different sections defined (meta data, conditions and substances). Every section holds a listing of entries with an id and a label. Within every entry one to multiple pre set values can be included. These are in the editor shown as a drop down box giving the user the possibility to select one value. Via the dictLocation attribute a dictionary is bound to this mapping file, and its entries being used to generate tooltips for the meta data entries.

Danksagung

Ich danke:

allen, die an dem Gelingen dieser Arbeit direkt und indirekt beteiligt waren, besonders denen, die hier nicht explizit aufgeführt sind.

PD Dr. C. Steinbeck für die Aufnahme in seine Arbeitsgruppe, die Bereitstellung des Themas und die ständige Bereitschaft zur Diskussion.

Prof. Dr. D. Schomburg und allen Mitglieder der Research Group for Molecular Informatics und des Cologne University Bioinformatics Center für die gute Zusammenarbeit und die angenehme Arbeitsatmosphäre.

Stefan Kuhn und Philipp Heuser für die gute Zusammenarbeit und die inspirierenden Diskussionen.

abschließend allen Korrekturlesern für ihre Mühe.

Erklärung

Ich versichere, dass ich die von mir vorgelegte Dissertation selbständig angefertigt, die benutzten Quellen und Hilfsmittel vollständig angegeben und die Stellen der Arbeit − einschließlich Tabellen, Karten und Abbildungen −, die anderen Werken im Wortlaut oder dem Sinn nach entnommen sind, in jedem Einzelfall als Entlehnung kenntlich gemacht habe; dass diese Dissertation noch keiner anderen Fakultät oder Universität zur Prüfung vorgelegen hat; dass sie − abgesehen von unten angegebenen Teilpublikationen − noch nicht veröffentlicht worden ist sowie, dass ich eine solche Veröffentlichung vor Abschluss des Promotionsverfahrens nicht vornehmen werde. Die Bestimmungen der Promotionsordnung sind mir bekannt. Die von mir vorgelegte Dissertation ist von PD Dr. Christoph Steinbeck betreut worden.

Tobias Helmus

Teilpublikationen:

Kuhn, S., Helmus, T., Lancashire, R., Murray-Rust, P., Rzepa, H., Steinbeck, C., Willighagen, E.: **Chemical Markup, XML, and the World Wide Web. 7. CMLSpect, an XML vocabulary for spectral data**. *Journal of Chemical Information and Modeling,* Submitted.

Spjuth, O., Helmus, T., Willighagen, E.L., Kuhn, S., Eklund, M., Wagener, J., Murray-Rust, P., Steinbeck, C. & Wikberg, J.ES.: **Bioclipse: an open source workbench for chemo- and bioinformatics**. *BMC Bioinformatics* 2007, **8**:59.

Köln, den 05.07.2007

Lebenslauf

Zur Person

Dipl. Biologe
Tobias Helmus
geboren am 26.05.1974 in Münster
Staatsangehörigkeit: deutsch

Zorndorfstraße 10
50737 Köln
Email: thelmus@web.de

Ausbildung

05.2004-06.2007	Wissenschaftlicher Mitarbeiter / Promotion zum Thema „Encoding, Storing and Searching of Analytical Properties and Assigned Metabolite Structures“ in der **Research Group for Molecular Informatics, Cologne University Bioinformatics Center (CUBIC), Universität zu Köln**
04.2003-04.2004	Aufbaustudium Bioinformatik am **Cologne University Bioinformatics Center (CUBIC)** – finanziert durch ein Stipendium des BMBF
02.2001-07.2002	Diplomarbeit mit dem Thema: „Interaktion verschiedener Cr-Spezies mit Immunzellen des Karpfens (*Cyprinus Carpio*) unter unterschiedlichen Kulturbedingungen“ im **Fachgebiet Fischkrankheiten, Zentrum für Infektionsmedizin der Tierärztlichen Hochschule Hannover**
09.1998-08.1999	Studium der marinen und terrestrischen tropischen Biologie an der **Universidad Nacional, Costa Rica,** als DAAD-Stipendiat im Rahmen eines IAS-Programms
10.1995-12.2002	Studium der Diplom-Biologie an der **Universität Hannover**
1987-1994	**St.Ursula-Schule (Gymnasium)** 30171 Hannover

Köln, den 05.07.2007