doi:10.2533/chimia.2012.132

CyBy^2: A Structure-based Data Management Tool for Chemical and Biological Data

Stefan Höck and Rainer Riedl*

*Correspondence: Dr. R. Riedl, ZHAW Wädenswil, Einsiedlerstrasse 31, CH-8820 Wädenswil, Tel.: +41 58 934 56 18, E-mail: rainer.riedl@zhaw.ch

Abstract: We report the development of a powerful data management tool for chemical and biological data: *CyBy^2*. *CyBy^2* is a structure-based information management tool used to store and visualize structural data alongside additional information such as project assignment, physical information, spectroscopic data, biological activity, functional data and synthetic procedures. The application consists of a database, an application server, used to query and update the database, and a client application with a rich graphical user interface (GUI) used to interact with the server.

Keywords: Medicinal chemistry · Project management · Scala programming language · Structure-activity relationship · Structure-based data management

Introduction

For the successful management of R&D projects it is mandatory to employ powerful data management tools. Multidisciplinary projects such as medicinal chemistry projects or material science projects rely in particular on such information management tools.[1] Structural data have to be combined with a variety of functional data in order to extract structure–activity relationships from massive data sets. Therefore, it is of outmost importance to have the means to query those data sets for structural information and to visualize the results in an efficient and convenient manner in order to make the best use of the scientific information.

Application Setup

CyBy^2 consists of three parts: An *Apache Derby*[2] database, used to store chemical structures alongside related information, a *RESTful* web service[3] that handles queries and modifications of the underlying database, and a rich client application that allows for convenient interaction of the users with the server (Fig. 1). As this application allows for the powerful analysis of data from chemistry and biology, the acronym *CyBy^2* has been chosen.

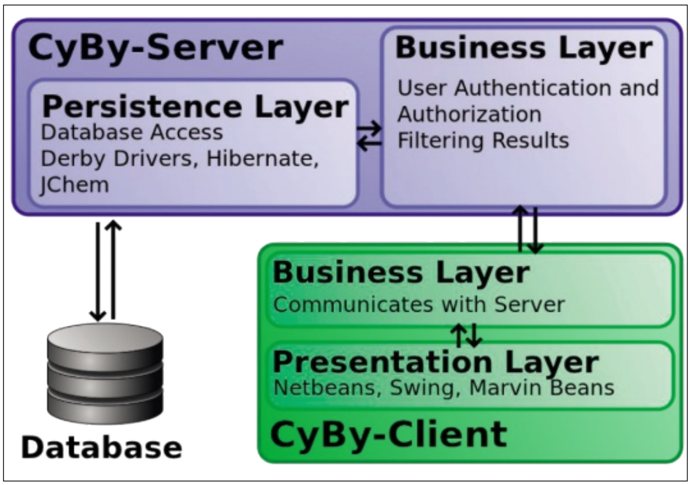

Fig. 1. Application layout.

The Language of Choice

The *CyBy^2* client and server are both written in *Scala*, a multiparadigm programming language developed by the Odersky group at the EPFL in Lausanne, Switzerland.[4] *Scala* offers several advantages over other programming languages: It is a fully object-oriented programming language with a strong support for the functional programming paradigm. The functional programming approach is especially well suited for scientific applications such as *CyBy^2*, and indeed, *CyBy^2* makes heavy use of *Scala's* functional programming constructs. On the other hand, *Scala* is compiled to *Java* byte code and runs on the *Java Virtual Machine* (JVM).[5] Therefore, *Scala* applications usually run on a large variety of computer platforms. Also, *Java* libraries such as the *JChem* API[6a] can easily be integrated in and accessed from *Scala* applications.

Third-party Plugins

CyBy^2 makes use of several third-party application programming interfaces (APIs) and libraries. *JChem Base* is used for structure searching and chemical database access and management.[6a] *Marvin* is used to implement the drawing, displaying and characterization of chemical structures.[6b] The GUI of the *CyBy^2* client as well as its update functionality and modular application layout are provided by the *NetBeans Platform*.[7] The *CyBy^2* server runs on a *GlassFish* Application Server.[8]

Database Setup

All data entered in *CyBy^2* are stored in a database whose overall layout is shown in Fig. 2. The central part of the database is the 'container' entity, usually a single flask with a defined physical location in our laboratories. This can either be a substance bought from a commercial supplier or research compounds synthesized in our group.

Containers have a many-to-one relationship with compounds, which are stored as chemical structures in the database. Each compound must either be a unique chemical structure or a unique name with an empty structure. We decided to allow for empty structures to be entered in the database since the database should not only contain small molecule compounds but also complex chemical structures such as polymers whose correct structure cannot always be determined.

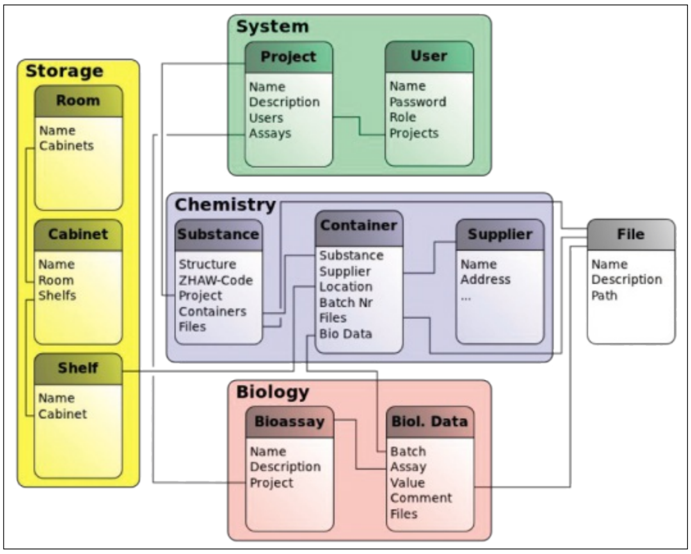

Fig. 2. Database layout.

Containers can be linked to functional data such as biological assay data. This allows *CyBy*^{2} to manage links between chemical structures and biological activity. Each biological data entry in the database is linked to an assay defining the experimental setup and biological target used. The database can easily be queried for all compounds that were tested for a given assay.

Each of the three types of entities – compounds, containers, and biological data – can be linked to files stored in pool folders on the network. This way, compounds can be linked to spectral data, batches (= containers) to synthetic procedures, and biological data to the corresponding experimental raw data.

Compounds and assays are both grouped in projects. These projects are used to add another layer of logical structure to the database as well as to restrict user access to confidential data. Each user in *CyBy*^{2} usually only has access to a subset of projects. This allows *CyBy*^{2} to be used as the primary information tool for an entire group of researches without creating intellectual property related issues.

Server Setup

The *CyBy*^{2} server has been designed as a *RESTful* web service.[3] It communicates with clients *via* http-requests, where each service provided by the server can be accessed *via* its own URL. The server is not only responsible for accessing and modifying the database, it is also used for user authentication and authorization as well as long-term calculations whose results can be memorized, to speed up additional computations.

With the actual design, the server loads the content of the whole database into memory upon startup. This approach makes data processing and querying very fast and facilitates memorization of heavily used query results. It also facilitates the computation of type-safe multi-table queries using the whole power of *Scala's* functional programming syntax.

Client Setup

The *CyBy*^{2} client is a rich client that is written in *Scala*[4] and built on the *NetBeans* Platform.[8] It uses *Marvin* to display and draw chemical structures.[7]

The client's main purpose is to display compound hit sets returned by the server after querying the database (Fig. 3). These hit sets are either displayed as tree-like structures, or as a grid of compounds in a modified table view. In both cases, each entry is represented as a node in *NetBeans*, which allows for the convenient linking of pop-up actions, icons, and additional functionality to an entry *via* the *NetBeans* API. Ranges and groups of nodes can be selected individually and exported as files of different formats, .sdf and .mol being the most often used. When exporting compounds, the user can choose to export additional data as .sdf tags alongside the structural information.

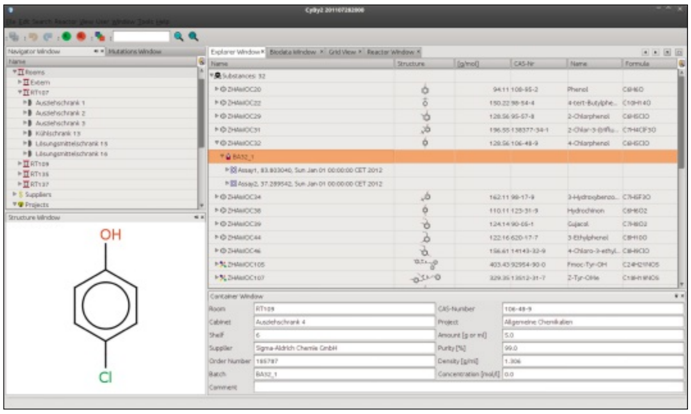

Fig. 3. Application main view.

Database Queries

A database application must support all kinds of combined queries to collect data from the database according to different criteria.

In a database containing chemical compounds, the search for substructures is of highest priority (Fig. 4). This is not a trivial task and there exist several APIs, both open source and commercial, for querying chemical databases. *CyBy*^{2} makes heavy use of *JChem Base* to query its database for substructures.

Text-based search is also important and often the quickest way when one knows what one is looking for. *CyBy*^{2} contains a quick search functionality that searches all textual fields in the database for the occurrence of a given regular expression[9] entered by the user. When a hit is found high up in the data tree – in the name of a project for instance – all compounds belonging to that node are returned by the server. If on the other hand the search string is only found in the name of a file linked to a container, only the container in question and its parent compound are returned. Additional containers and files of the parent compound and container are not displayed. This allows for the precise retrieval of the desired data.

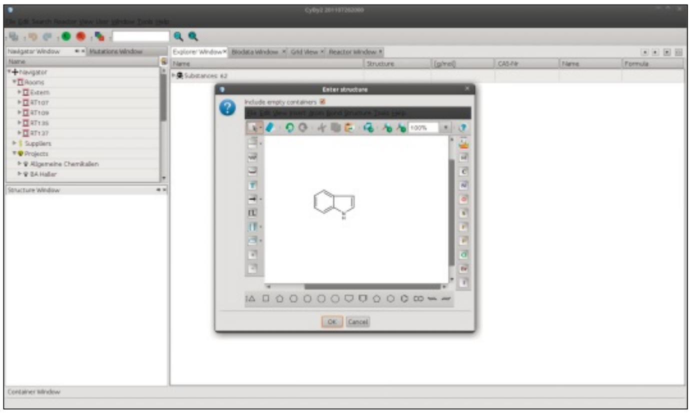

Fig. 4. Substructure search.

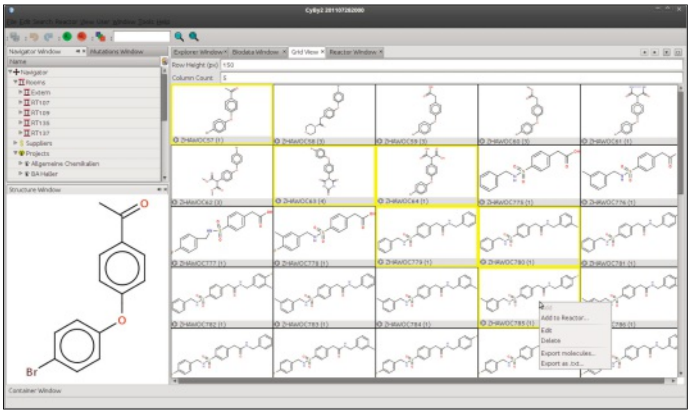

Fig. 5. Grid view.

The database can also be queried by supplier, assay, project, or location (room, cabinet, or shelf). Most of these queries allow for the selection of more than one node in the user interface. The user can for instance select a couple of assays and display immediately all biological data linked to those assays.

Hit Set Visualization

Hits from database queries can be visualized in different views. The main view is a *NetBeans* OutlineView that displays hits as tree-like structures. The top nodes are compounds that are linked to several containers and files. Containers contain biological data and again files as subnodes, and each piece of biological data can also be linked to several files. This allows for a logical view of compounds and their related data that mirrors the layout of the underlying database.

In many cases, one wants to get a quick overview of the compounds returned by a query. For this case, *CyBy*^{2} provides a grid view, which displays the structures of compounds in a grid of adjustable size, together with only a small set of additional data. Similar to the OutlineView, compounds in the grid view can be selected individually for further processing (Fig. 5).

In a medicinal chemistry environment the correlation between compound structure and functional biological data is of special importance. *CyBy*^{2} is designed to display a collected set of biological data for a selection of compounds. This data is calculated upon demand by the client and consists of a table view that displays each compound in a row and each assay in a column. To facilitate the analysis of functional biological data, *CyBy*^{2} supports user-defined conditional formatting across the entire data set of the biological data view (Fig. 6).

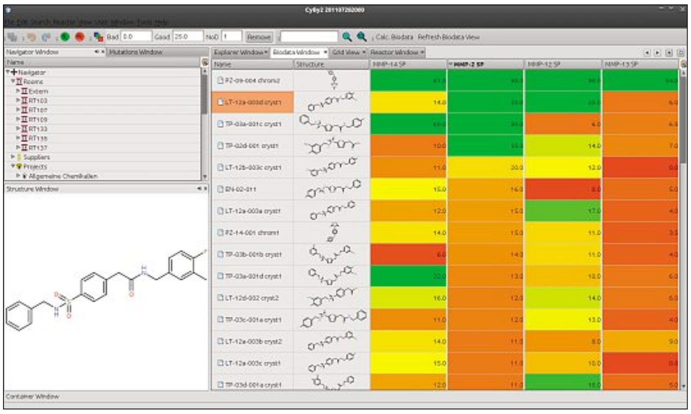

Fig. 6. Biological data view with conditional formatting.

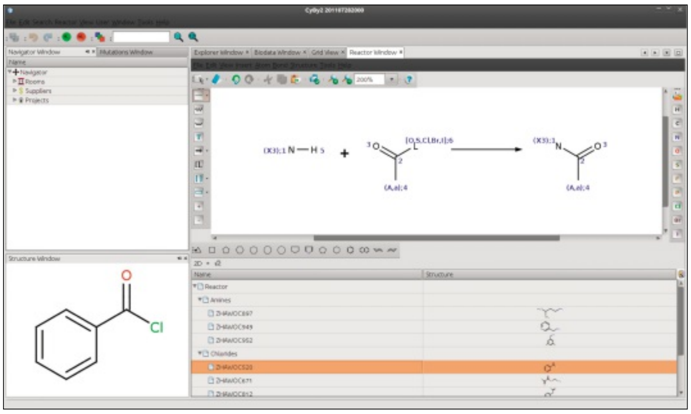

Fig. 7. Reactor implementation used for library enumeration.

Conclusion

We have developed a powerful structure-based data management tool for handling complex data sets. The application is very user friendly and can be used by research groups for managing their entire project portfolio originating from different scientific disciplines. Due to the modular setup of *NetBeans* Platform applications, *CyBy*^{2} can easily be extended with additional functionality. Only recently, we implemented *Chemaxon's* Reactor functionality[6c] that can be used for library enumeration based on a given chemical reaction (Fig. 7). The *CyBy*^{2} database can be queried for reactants which can then be combined to create new compound libraries. Other functional data of different scientific origin can be easily added to the database in order to use this information tool for the successful steering of the respective project. Currently, we are extending *CyBy*^{2} in order to use it for the storage and analysis of molecular modeling experiments performed in our medicinal chemistry projects.

Acknowledgements

This project was financially supported by the ZHAW (Development of an integrated database tool for the management of medicinal chemistry projects). We appreciate the help of and the fruitful discussions with internal and external users for the benefit of the software.

Received: January 18, 2012

[1] Molecular Medicine Tri-Conference 2012, 'Fourth Annual Integrated R&D Informatics & Knowledge Management', San Francisco, February 21–23, **2012.**

[2] *Apache* Software Foundation, 'Derby Reference Manual', Version 10, **2006**, *http://db.apache.org*.

[3] a) W. J. Burke, '*RESTful Java* with JAX-RS', O'Reilly Media Inc., Sebastopol, **2010**; b) S. Allamaraju, '*RESTful* Web Services Cookbook', O'Reilly Media Inc., Sebastopol, **2010**.

[4] M. Odersky, P. Altherr, V. Cremet, I. Dragos, G. Dubochet, B. Emir, S. McDirmid, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman, L. Spoon, M. Zenger, 'An Overview of the Scala Programming Language', EPFL, Lausanne, **2006**. For a thorough introduction see: M. Odersky, L. Spoon, B. Venners, 'Programming in *Scala*', Second Edition, Artima Press, Walnut Creek, **2010**.

[5] *www.java.com*. Language specification: J. Gosling, B. Joy, G. Steele, G. Bracha, 'The *Java* Language Specification', Third Edition, Addison-Wesley, Santa Clara, **2005**.

[6] *ChemAxon* API (*http://www.chemaxon.com*): a) *JChem Base*, 5.5.0, **2011**; b) *Marvin* 5.5.0, **2011**; c) Reactor 5.5.0, **2011**.

[7] *http://platform.netbeans.org*. For an introduction see: H. Böck, 'The Definitive Guide to *NetBeans* Platform 7', Apress, New York, **2011**.

[8] *http://glassfish.java.net*. For an introduction see: A. Goncalves, 'Beginning *Java*EE6PlatformwithGlassFish3',SecondEdition,Apress,NewYork,**2010**.

[9] A regular expression is a means to match a string against a given pattern such as the occurrence of a single character, a whole word, or a more complex pattern including wild cards and placeholders. Regular expressions are very powerful but their syntax is not trivial. For an introduction see: a) *http://www.regular-expressions.info*; b) J. Goyvaerts, S. Levithan, 'Regular Expressions Cookbook', First Edition, O'Reilly Media Inc., Sebastopol, **2009**.