

jectIOListener interfaces, this class has the function of containing information about the IO setting, and store the active value.

CDK currently has four **IOSetting** implementations: **BooleanIOSetting**, **IntegerIOSetting**, **StringIOSetting** and **OptionIOSetting**. The first three classes can only take specific settings: yes or no, integers, and **Strings** respectively.

The **OptionIOSetting** is a special class; besides one default, it contains a list of suggestions. The **ChemObjectIOListeners**, i.e. all current implemen-

tations, will offer this list of options to the user, as depicted in Fig. 1.

If you plan to add **IOSettings** to existing or new IO classes, please have a look at the source code of the **GaussianInputWriter** or the **CMLWriter**.

I hope you appreciate the simplicity of the whole architecture.

Egon Willighagen

Radboud University Nijmegen, The Netherlands

e.willighagen@science.ru.nl

First steps in the implementation of a force field for the CDK package

Development of a 3D-structure model builder tool.

by *Christian Hoppe*

Introduction

As mentioned in the CDK News 1.1[1] we recently started to work on a CDK implementation of a force field. Force fields are widely used in the area of chemoinformatics, e.g. to optimize the 3D geometry of a molecule, in conformational search or studies in molecular dynamics[2]. A force field of a molecule describes with a set of equations the potential energy surface of that molecule. Since those equations are not based on first principles empirically or quantenmechanically precalculated data is needed. This parameterisation limits the application of force fields and over the years several different sets of equations with corresponding data have been developed[2, 3]. In general one can differentiate these force fields in two classes, one for macromolecules and one for small organic molecules (usually less than 200 heavy atoms)[2]. We are aiming at the latter and plan to provide at least a couple of minimization algorithms which can be used in combination with a force field to optimize the 3D-structure of a molecule.

3D-Model builder

In the process of implementing a force field method for the CDK package, we started with the development of a tool for the generation of reasonable 3D starting structures. One of the most severe problems in the generation of 3D coordinates is the layout of rings and ring systems. Therefore we followed a knowledge-based approach in collecting and storing unique ring systems (ignoring different confor-

mations) to use them as templates in the 3D structure generation process[4]. This procedure differs from real 3D structure generating programs as e.g. CORINA[5], which stores for small ring systems conformational templates. We did not consider conformations at this step, because we only want to generate a reasonable starting structure for the force field. We downloaded a collection of small molecules as molfile from the nci databank (<http://cactus.nci.nih.gov/ncidb2/download.html>)[6]. To extract the molecule data stored in this file (249,071 3D-structures) the **IteratingMDLReader** from the CDK software package was used.

```
import org.openscience.cdk.io.iterator
    .IteratingMDLReader;

iteratingMdlReader = new IteratingMDLReader(
    new BufferedReader(new FileReader(nci_molecules))
);

Molecule molecule = null;
while (iteratingMdlReader.hasNext()) {
    molecule = (Molecule)imdl.next();
    // ...
}
```

To identify ring systems in a molecule one can use e.g. the **SSSRFinder** class. This class identifies the smallest set of smallest rings of a molecule. The **RingPartitioner** class can partition this **sssRingSet** into **RingSets** of connected rings which share at least an atom, a bond or three or more atoms with another ring in the **RingSet**. The container class **RingSet** is able to store, handle and manipulate rings and ring systems.

```
import org.openscience.cdk.RingSet;
import org.openscience.cdk.ringsearch.SSSRFinder;
import org.openscience.cdk.ringsearch
    .RingPartitioner;

SSSRFinder sssrf = new SSSRFinder();
```

```
Vector ringSets = RingPartitioner.partitionRings(
    sssrf.findSSSR(molecule)
);
```

After a scan of all 249,071 NCI molecules, we collected 11610 unique ring systems. In order to build a 3D structure for a new modeling candidate, we first examined its molecular structure for the existence of one of the template ring systems. To map the template ring systems onto a query structure the **UniversallIsomorphismTester** class should be used. With the **QueryAtomContainer** class one is now able to configure a substructure or isomorphism search for your own requirements (e.g. search only for equivalent bond systems). If a template ring system could be mapped, its coordinates were assigned to the modeling candidate and aliphatic chains were layed out thereafter.

Currently, molecules with unknown ring systems cannot be handled by this approach.

Dr. Christian Hoppe
 Universität zu Köln, Germany
 christian.hoppe@uni-koeln.de

Bibliography

- [1] E.L. Willighagen. What's 2004 going to bring? *CDK News*, 1(1):3–4, 2004.
- [2] N.L. Allinger. *Force Fields: A Brief Introduction*, volume 2 of *Encyclopedia of Computational Chemistry*, pages 1013–1015. 1998.
- [3] J.R. Maple. *Force Fields: A General Discussion*, volume 2 of *Encyclopedia of Computational Chemistry*, pages 1015–1024. 1998.
- [4] J. Sadowski. *3D Structure Generation*, volume 1 of *Handbook of Chemoinformatics*, pages 231–261. 2003.
- [5] Rudolph C. Sadowski J. Gasteiger, J. *Automatic Generation of 3D-Atomic Coordinates for Organic Molecules.*, volume 3 of *Tetrahedron Comput. Methodol.*, pages 537–547. 1990.
- [6] Takahasi Y. Abe H. Sasaki S. Ihlenfeldt, W.D. *CACTVS: A Chemistry Algorithm Development Environment*, pages 102–105. Daijuukagakutouronkai Dainijuukai Kouzoukassiseisoukan Shinpojiumu Kouenyoushishuu. 1992.

Spok - The Spectrum Organisation Kit

An introductory overview on the current state of the Spectrum Organisation Kit.

by Tobias Helmus

The Spectrum Organisation Kit is a program written in Java for the organisation and visualisation of spectral data. It is now in an alpha state and is intended to be used by experimental scientists for the analysis of spectral data, the assignment of these data to structural information and the management of larger amounts of spectral/structural data. It will be published under an open source license.

At the moment the program is able to display continuous and peak data in form of charts by using the JFreeChart library [1]. A comparable view of the peak and the continuous chart is provided, and zooming into all charts is already implemented. For giving the possibility to enter a structure for every spectrum the JChemPaint editor for 2D molecular structures [2] is embedded into the application.

Persistence of the spectral and structural data is realised by serialisation of the spectrum objects to XML files via the XStream library [3], whereas the structural information is written to CML files using the CDK **CMLWriter**. The code for the latter looks like this:

```
FileWriter out = new FileWriter("Filename");
```

```
CMLWriter cmlwriter = new CMLWriter(out);
cmlwriter.write((SetOfMolecules) structure);
cmlwriter.close();
```

where structure is the **SetOfMolecules** constructed by painting a structure in JChemPaint [2].

For the spectrum object, the `toXML()` method of the XStream package returns the java object in one XMLString which then can be written to a file with a standard **FileWriter**:

```
File file = new File("Filename");
FileWriter out;
XStream xstream = new XStream();
String xml = xstream.toXML((Spectrum) spectrum);
out = new FileWriter(file);
out.write(xml);
out.close();
```

At the time of writing the program was capable of reading spectral data in the jcamp-dx format. For this purpose the jcamp-dx library hosted on SourceForge [4] was used, which will be the reference implementation of the IUPAC JCAMP-DX spectroscopy data standard [5]. This library offers routines for reading and writing all spectrum types and data formats defined in this standard. Using the **JCampReader** is very straightforward and can easiest be done by creating a spectrum from a string containing the whole jcamp-dx file:

```
JCampReader jcamp = JCAMPReader.getInstance();
jcampSpectrum = jcamp.createSpectrum(jcampString);
```