RESEARCH

The Chemistry Development Kit (CDK). 3. Atom typing, Rendering, Molecular Formula, and Substructure Searching

Egon L Willighagen[1*], John W May[2], Jonathan Alvarsson[3], Arvid Berg[3], Nina Jeliazkova[4], Tomáš Pluskal[??], Miguel Rojas-Cherto[??], Ola Spjuth[3], Gilleain Torrance[??], Rajarshi Guha[5] and Christoph Steinbeck[6]

[*]Correspondence: egon.willighagen@maastrichtuniversity.n
[1] Dept of Bioinformatics - BiGCaT, NUTRIM, Maastricht University, NL-6200 MD, Maastricht, The Netherlands
Full list of author information is available at the end of the article

Abstract

Background: Cheminformatics is a well-established field with many applications in chemistry, biology, drug discovery, and others. The Chemistry Development Kit (CDK) has become a widely used Open Source cheminformatics toolkit, providing various models to represent chemical structures, of which the chemical graph is essential. However, in the first five years of the project increased so much in size that interdependencies between components grew unmanageable large, resulting in unpredictable instabilities.

Results: We here report improvements to the CDK since the 1.2 release series made to accommodate both the increased complexity of the library, as well as significant improvements of and additions to the functionality of the library. Second, we outline how the CDK evolved with respect to quality control and the approach we have adopted to ensure stability, including a peer review mechanism. Additionally, a selection of the new APIs that have been introduced will be discussed: atom type perception, substructure searching, molecular fingerprints, rendering of molecules, and handling of molecular formulas.

Conclusions: With this paper we have shown the continued effort to provide a free, Open Source cheminformatics library, and show that such collaborative projects can exist over a long period. We have taken advantage from the community support, and show that an open source cheminformatics project can act as a peer reviewed publishing platform for scientific computing software.

Keywords: Java; cheminformatics; bioinformatics

Background

Open Source cheminformatics has made significant steps forward recently [1]. The Chemistry Development Kit (CDK) is one of the tools under the Blue Obelisk wings, and previously documented version have been widely adopted [2, 3]. For example, new tools expose CDK functionality, such as Cinfony [4], rcdk [5], and workflow plugins for Taverna [6] and KNIME [7]. Furthermore, projects with specific functionality have emerged building on the CDK, including jCompoundMapper [8], ScaffoldHunter [9], OMG [10], Padel [11], AMBIT-SMARTS [12], AMBIT-Tautomer [13], and SMSD [14, 15]. Other tools used previous versions already, and follow newer releases of the CDK, such as Bioclipse [16, 17] and AMBIT [18]. While the above tools suggest that the CDK is primarily used to create general tools, it is important

to note that it is also used to solve specific questions, like finding the maximally bridging rings in chemical structures [19].

However, previous version of the CDK were written by a community with specific applications in mind, of which structure elucidation was one. It could therefore be rather competitive in some cheminformatics domains, such as fingerprinting [20, 21], in other parts it could be inadequate, such in the capturing of stereochemistry.

Continued community development of the CDK in the last few years, however, has shown that Open Source software is here to stay. The adoption of automatic build systems and quality control methodologies such as unit testing, automated source code validation, and peer review by fellow developers have greatly improved the stability of the library, At the expense of slowing down the development, it has allowed for cleaning up interdependencies between modules of code, and has reset the scalability of the development model, allowing for a lot of new functionality in code APIs while maintaining the quality of code depending on those core APIs.

This has resulted in important improved functionality, such as a separately published InChI functionality [22] and greatly improved ring detection functionality [23], but also a core atom type perception module, covering a much wider set of elements than previous CDK versions and capturing many more charged and radical atom species, a more comprehensive fingerprinting API, many speed and stability improvements, and more.

Results and Discussion

New APIs

We here outline various new and improved APIs in the CDK library since the two previous publications.

Atom Typing

Atom type perception is a core cheminformations functionality: the atom types describe chemical features of atoms, such as the number of neighbors, possible formal charges, (approximate) hybridization, electron distribution over orbitals, etc. However, the CDK commonly integrated atom type perception into algorithms, resulting in multiple and diverging copies of similar atom type schemes. With the need to add many new atom types for previously uncovered element, but also charged and radical atom types, a new approach was needed.

This CDK version has isolated a new atom typing framework, removing the perception of atom types from various algorithms, allowing the perception to be tested separately. The new code defines the atom types using a list that specifies for each type the element symbol, hybridization, formal charge, number of lone pairs, and an enumeration of the bond orders (see Figure 1). This list of properties captures the information needed for the various algorithms in the CDK. For example, hybridization information can be used in certain aromaticity models (see later), and the lone pair information is needed for resonance structure calculation needed, for example, for Gasteiger π-charges.

A reference perception class, CDKAtomTypeMatcher, has been written that perceives these atom types, and validates the perception automatically against the properties defined by the ontology. This class handles a variety in types of missing information, as commonly resulting from various (file) formats; for example, it

can handle undefined hydrogen counts and undefined double bond positions if hybridization information is provided instead. That makes the code complex, and at times slow, but so far shown sufficiently performant for many applications. Possibly, in the future alternative algorithms may be implemented, while that would again increase the maintenance burden.

Stereochemistry

The new CDK version introduces a new API for stereochemistry information: the interfaces IStereoElement was introduced with the implementations TetrahedralChirality, for four coordinate compounds like bromo-chloro-fluoro-iodomethane, DoubleBondStereochemistry, for compounds like trans-2-butene, and ExtendedTetrahedral, for compounds like allene.

An atom container has a method to iterate over all stereo elements:

```
for (IStereoElement element : container.) {
  if (element instanceof TetrahedralChirality) {
    // ...
  } else if (element instanceof DoubleBondStereochemistry) {
    // ...
  } else if (element instanceof ExtendedTetrahedral) {
    // ...
  }
}
```

Signatures

An implementation has been provided of the Signature structure descriptor for molecules [24]. These act as a linear notation - like the SMILES format - for the whole molecule as well as for connected substructures rooted at a single atom. The descriptor can also be canonicalized to provide isomorphism-independent representations [25].

Rendering API

A new rendering API has been introduced to make the rendering code independent from Java widget toolkits. The previous code was tightly linked to the Swing toolkit, but other tools use different widget toolkits. For example, Bioclipse is based on Eclipse which uses the SWT [16].

A second new design goal was introduced to balance between size restrictions of some use cases, such as Java applets, and the rendering functionality. In particular, some functionality, even after Modularization, needed considerable parts of the CDK library, making creation if a small-sized applet unfeasible. Therefore, the rendering API was modularized to allow splitting up rendering functionality into modules, with varying CDK dependencies.

Moreover, a simplified API has been introduced that addresses most of the common rendering needs, with the DepictionGenerator class. To depict benzene the following code can be used:

```
new DepictionGenerator()
  .withSize(300, 300)
```

```
  .depict(benzene)
  .writeTo("BenzeneLocalized.png");
```

Many of the rendering options are available as parameters in the core API and as methods on the DepictionGenerator class. This includes substructure coloring, exemplified with this example reaction (see Fig. 2):
TODO

Structure Diagram Layout
The structure diagram layout has been rewritten and the new code solves a number of long standing issues. Particularly, collision detection has been greatly improved. Figure 3 shows a difference in output between the old code base, with and without overlap resolving, and with the new code. The new code can be used like:
TODO

Molecular Formula
The chemical formula is the basic/simple chemical representation of a compound. It defines the number of isotopes or elements that constitute the composition of a compound without describing how atoms are bonded. With the rise of metabolomics it has become increasingly relevant to have full support for these in cheminformatics libraries [26].

The CDK interfaces can handle several concepts related to chemical formulas: the formula itself, sets of formulas, chemical formula ranges, adducts, isotope containers and patterns, and rules to filter formula sets. These new tools can be used for a number of tasks, including calculate the isotopic pattern from a given chemical formula, determining the possible elemental compositions for a given mass, and calculate the exact mass from a given chemical formula.

The following code calculates all possible chemical formula for a given accurate mass, within some allowed counts for each element:

```
Isotopes ifac = Isotopes.getInstance();
MolecularFormulaRange range =
  new MolecularFormulaRange();
range.addIsotope( ifac.getMajorIsotope("C"), 8, 20);
range.addIsotope( ifac.getMajorIsotope("H"), 0, 20);
range.addIsotope( ifac.getMajorIsotope("O"), 0, 1);
range.addIsotope( ifac.getMajorIsotope("N"), 0, 1);

MolecularFormulaGenerator tool =
  new MolecularFormulaGenerator(
    SilentChemObjectBuilder.getInstance(),
    133.0, 133.1, range
  );
IMolecularFormulaSet mfSet = tool.getAllFormulas();
for (mf in mfSet) {
  println MolecularFormulaManipulator.getString(mf) + " " +
    MolecularFormulaManipulator.getTotalExactMass(mf)
}
```

This gives the following output:

```
C11H 133.007825032
C9H11N 133.089149352
C9H9O 133.065339908
C8H7NO 133.052763844
```

SMILES parser and generator

The SMILES parsing has been replaced code in the external Beam project [27]. This BSD-licensed SMILES parser is a complete implementation of the SMILES and OpenSMILES (http://opensmiles.org/) specifications by one of the authors (including stereochemistry), and is independent of the CDK library. The SmilesParser API uses this library underneath, and the Beam API is hidden by this class. The most significant change here is that the SMILES parser now automatically locates the positions of double bonds, but this can be turned off:

```
SmilesParser parser = new SmilesParser(
  SilentChemObjectBuilder.getInstance()
);
parser.kekulise(false);
mol = parser.parseSmiles(smi);
```

The SMILES generation API has also been simplified and made more flexible. Creating unique SMILES or aromatic SMILES is now done by first creating a SMILES generator instance which can make generic or unique SMILES, and with or without simplifying aromatic rings using the lower case element symbols for the organic subset. This generator can then be repeatedly called:

```
uniqueGenerator = SmilesGenerator.unique()
aromaticGenerator = SmilesGenerator.generic().aromatic()
smiles = uniqueGenerator.createSMILES(mol)
smiles2 = atomaticGenerator.createSMILES(mol)
```

Ring finding

Ring finding is a key functionality in a cheminformatics library, and the CDK knows a long history of ring finding [23]. Particularly, non-redundant ring sets have seen particular interest, such as the smallest set of smallest rings, for which the CDK implements two classical algorithms [28, 29]. Recent work by one the authors has implemented a new, faster algorithm, allowing searching for various types of (non-redundant) ring sets [23]. These are available via the new Cycles API:

```
allCycles = Cycles.all(mol)
relevantCycles = Cycles.relevant(mol)
essentialCycles = Cycles.essential(mol)
sssrCycles = Cycles.sssr(mol)
```

Aromaticity

Aromaticity has seen many definitions in the past and for cheminformatics it frequently is algorithmically defined. The outcome of an aromaticity calculation depends on a number of atom type features and heuristics, which in literature often are not well defined. Based on this information several different algorithmic definitions

of aromaticity can be defined. Older CDK versions had various models implemented and the code was scattered throughout the library. This was unified resulting in a three models, of which two are based on the CDK atom typer, to provide the number of electrons donated by each atom to the delocalized π system. The difference between these two models is how double bonds pointing outwards from a ring are contributing.

The current CDK version further generalizes the idea that aromaticity is a model, and provides an API that allows you to select out of various options providing greater interoperability with other toolkits. The new Aromaticity class allows to build a custom model by selecting and combining options. For example, to reproduce the functionality of the previous CDKAromaticity class:

```
Aromaticity aromaticity = new Aromaticity(
  ElectronDonation.cdk(), Cycles.cdkAromaticSet()
);
```

Here, the CDK model for counting donated electrons is used, as the ring systems the CDK previously included in aromaticity counting, which was limited in the number of fused rings systems. However, an alternative aromaticity calculator that does consider all possible ring systems can now be easily created with:

```
Aromaticity aromaticity = new Aromaticity(
  ElectronDonation.cdk(), Cycles.all()
);
```

New Builders

Originally, the CDK was developed as a shared library between JChemPaint and Jmol. The former used a MVC approach with an event-passing mechanism to update the view when the model was changed. This can cause an cascade of change events being passed around. To address, interfaces were introduced allowing multiple implementations of the core interfaces. The IChemObjectBuilders play an important role here, allowing implementations of the interfaces to be instantiated without the need of explicitly referencing those implementations.

However, the CDK 1.0 and 1.2 implementations of the IChemObjectBuilder had one method for each constructor, resulting in very large interface. Moreover, the API changed whenever a new class was introduced, and existing methods changed when constructors were updated. To simplify the API, the new IChemObjectBuilder collapsed all methods returning new implementations into a single method, which takes as a first parameter the class of the interface that is wished to be constructed. All further parameters are passes as parameter to the class constructor.

For example, to construct a new atom from its element symbol, one would now write:

```
IChemObjectBuilder builder; // previously defined
IAtom atom = builder.newInstance(IAtom.class, "C");
```

Increasingly, the CDK library is now written against the interfaces, and when new instanced are needed, these builders are being used. This allows to run a certain CDK-based application with a specific builder, aimed at a particular use case. The original implementation is available via the DefaultChemObjectBuilder, which creates classes that pass around change events, just like the original CDK data

classes. However, a SilentChemObjectBuilder has been introduced that does not pass around change events, which makes code run about 10-20% faster. The third builder is the DataDebugChemObjectBuilder which generates debug information for all changes to the content of the data classes. This can be useful for debugging and other forms of code inspection.

Molecular fingerprints
Also the molecular fingerprints in CDK has seen development since the 1.2 release. Traditionally CDK fingerprints were represented using the *BitSet* class shipping with Java. The benefits of that class include such things as already implemented methods for modifying bit sets using other bit sets. However the class keeps a vector of bits in memory. The solution was excellent for hashed, relatively small fingerprints, *e.g.*, 1024 bits, *i.e.*, 10 bits indexing space. However implementing a fingerprint designed to avoid collisions with, *e.g.* a 32 bit indexing space using this approach would be highly memory-inefficient. To allow for multiple fingerprint representation implementations a bit fingerprint interface was introduced. Also, although fingerprints traditionally are bit vectors a count fingerprint was also introduced making fingerprints based on integer vectors supported in CDK as well. The fingerprints currently existing in CDK are listed in Table 1.

Improved Coding Standards

As the library grew over the years, so did the maintenance become more complex. Increasingly, the main branch did not compile, and bug fixing become increasingly difficult, as fixing a bug in one part of the code, broke some other code which made the wrong assumption about the first code.

To address these issues, we have adopted a number of coding standards. By no means there are meant to implement the best practices of source code development; instead, they attempt to find a balance between increasing code maintainability and being flexible enough to allow efficient code development. However, we appreciate the subjective nature of this statement, and some adopted guidelines have been heavily discussed in the community. The next sections describe some approaches the project have adopted that allows us to maintain the CDK library as it is today.

However, perhaps the biggest factor in improved code quality is that we instantiated a peer review process where any functionality changing patch is required to be reviewed by one independent, senior CDK developer for the development branch, and two reviewers for stable branches. This patch development system is supported by a number of automated validations steps as outlined below.

Modularization
One of the key approaches we have adopted, is to make the CDK more modular. The CDK assigns every class to a module, and defines dependencies between modules. For example, core modules are not allowed to depend on a module holding data classes implementing the CDK interfaces; instead, they may only depend on the interfaces themselves. This, for example, is to ensure that dependencies are minimized and to make it easier to exclude CDK functionality with third-party dependencies that are not needed.

An overview of key modules with description, important changes, and dependencies on third-party libraries is given in Table 2 and the dependencies between the CDK modules are depicted in Figure 4.

Documentation
The quality of JavaDoc of the CDK was originally tested with DocCheck, and later replaced by a custom written tool called OpenJavaDocCheck. With the move to Maven (explained elsewhere) which does not have integration for this tool, we adopted CheckStyle (http://checkstyle.sourceforge.net/). This tool reports about missing documentation and on documentation which is not properly annotated in the Java source files.

Testing
Years of development of the CDK library has resulted in a large suite of tests of various kinds. This include unit tests, which test core APIs, and functional testing, which test higher level functionality of the CDK. The latter include tests if algorithm implementations calculate the expected values, but also contain integrated tests, which involve more than one algorithms, such as SMILES parsing.

Code Quality
The project continues to use PMD (urlhttp://pmd.sf.net/) for code quality checking, but deviates from the default rules. For example, we are more liberal with variable name length. Moreover, a number of additional PMD tests have been developed specifically for the CDK, that, for example, test if a class uses the core interfaces instead of implementations of those interfaces. For example, that the code uses IAtom instead of Atom. That said, these tests do generate a few false positives, as the tests check the class name only, and not the Java package the class is in.

Git, branching, and patches
Another change made over the past years is the move to a Git-based version control system. Advantages the projects makes advantage of is the distributed nature of Git repositories, and easier branching and making available of patches. GitHub (https://github.com/cdk/cdk) has replaced SourceForge as the main source code hosting service where we can use novel approaches for commenting on code, pull requests, etc. These new features simplify our code review process.

Binary distributions

Maven packages
Another recent change is the choice of build system: we have moved away from Ant to Maven to deal with dependencies and compile and test the library. To keep the original idea of CDK modules, this move required to split up the source code in multiple source folders. Fortunately, modern integrated development environments have no trouble handling this, removing the original argument to have everything in one source folder.

On the other hand, for many modules, the test code is now more closely closely linked to the code being tested: both reside in the same folder, though we adhere

to the Maven custom to have src/main/java and a src/test/java folders. For a few modules, however, this solution introduces circular dependencies, in which case a separate Maven module is created for the tests.

The Maven packages for the CDK are available from Maven Central, which makes it easy for other projects to use. The full library can be included in other software by depending on the cdk artifact (http://mvnrepository.com/artifact/org.openscience.cdk/cdk) but dependencies can also be defined on individual CDK modules.

OSGi bundles

OSGi bundles are available for the CDK too, which are used by Bioclipse and KNIME.

Conclusions

Over the past years since the last CDK publication in 2006, a lot has changed. The functionality has seen many additions, and the stability of the development model has significantly improved. This paper outlines how peer review and quality control contributed to a much wider adoption of our cheminformatics library and seen many new additions. With 93 contributors, the CDK is alive and kicking.

However, there are many unsolved issues. Performance is perhaps one of the most important ones. The disadvantage of having an API that supports many data models, is that the API is heavy and the interface implementations hard to optimize.

Availability and requirements

- **Project Name**: The Chemistry Development Kit
- **Project home page**: https://github.com/cdk/cdk
- **Operating system(s)**: Windows, GNU/Linux, OS/X
- **Programming language**: Java
- **Other (optional) requirements**: JNI-InChI, Vecmath, Beam, Guava, JGraphT, Signatures, CMLDOM, XOM, JavaCC
- **License**: LGPL v2.1 or later
- **Any restrictions to use by non-academics**: None additional

Competing interests

JWM and NJ work for a company that sell solutions based on the CDK.

Authors contributions

All authors wrote and contributed source code or documentation to the CDK library. Some authors have peer-reviewed source code for the library. ELW, JWM, RG, and CS are project leaders. All authors have contributed to the content of this paper and approved the final version.

Acknowledgements

The authors acknoledge the great number of people who have contributed smaller and larger contributions to the CDK library. A full list of contributors is found in the AUTHORS file [30].

Author details

[1] Dept of Bioinformatics - BiGCaT, NUTRIM, Maastricht University, NL-6200 MD, Maastricht, The Netherlands. [2] ??, UK. [3] Department of Pharmaceutical Biosciences, Uppsala University, 751 24, Uppsala, Sweden. [4] Ideaconsult Ltd, A. Kanchev 4, 1000, Sofia, Bulgaria. [5] NIH Center for Translational Therapeutics, 9800 Medical Center Drive, MD 20878, Rockville, USA. [6] Chemoinformatics and Metabolism team, European Bioinformatics Institute, Hinxton, UK.

References

1. O'Boyle, N., Guha, R., Willighagen, E., Adams, S., Alvarsson, J., Bradley, J.C., Filippov, I., Hanson, R., Hanwell, M., Hutchison, G., James, C., Jeliazkova, N., Lang, A., Langner, K., Lonie, D., Lowe, D., Pansanel, J., Pavlov, D., Spjuth, O., Steinbeck, C., Tenderholt, A., Theisen, K., Murray-Rust, P.: Open Data, Open Source and Open Standards in chemistry: The Blue Obelisk five years on. Journal of Cheminformatics **3**(1), 37 (2011)
2. Steinbeck, C., Han, Y., Kuhn, S., Horlacher, O., Luttmann, E., Willighagen, E.: The Chemistry Development Kit (CDK): an open-source Java library for Chemo- and Bioinformatics. J Chem Inf Comput Sci **43**(2), 493–500 (2003)
3. Steinbeck, C., Hoppe, C., Kuhn, S., Floris, M., Guha, R., Willighagen, E.L.: Recent developments of the chemistry development kit (CDK) - an open-source java library for chemo- and bioinformatics. Current Pharmaceutical Design **12**(17), 2111–2120 (2006)
4. O'Boyle, N.M., Hutchison, G.R.: Cinfony - combining open source cheminformatics toolkits behind a common interface. Chemistry Central Journal **2** (2008)
5. Guha, R.: Chemical informatics functionality in r. Journal of Statistical Software **18**(5), 1–16 (2007)
6. Truszkowski, A., Jayaseelan, K.V., Neumann, S., Willighagen, E.L., Zielesny, A., Steinbeck, C.: New developments on the cheminformatics open workflow environment CDK-Taverna. Journal of Cheminformatics **3**(1), 1–10 (2011)
7. Beisken, S., Meinl, T., Wiswedel, B., de Figueiredo, L., Berthold, M., Steinbeck, C.: KNIME-CDK: Workflow-driven cheminformatics. BMC Bioinformatics **14**(1), 257 (2013)
8. Hinselmann, G., Rosenbaum, L., Jahn, A., Fechner, N., Zell, A.: jCompoundMapper: An open source java library and command-line tool for chemical fingerprints. Journal of Cheminformatics **3**(1), 3 (2011)
9. Wetzel, S., Klein, K., Renner, S., Rauh, D., Oprea, T.I., Mutzel, P., Waldmann, H.: Interactive exploration of chemical space with scaffold hunter. Nature chemical biology **5**(8), 581–583 (2009)
10. Peironcely, J.E., Rojas-Chertó, M., Fichera, D., Reijmers, T., Coulier, L., Faulon, J.-L., Hankemeier, T.: OMG: open molecule generator. Journal of Cheminformatics **4**(1), 1–13 (2012)
11. Yap, C.W.: PaDEL-descriptor: An open source software to calculate molecular descriptors and fingerprints. Journal of Computational Chemistry **32**(7), 1466–1474 (2011)
12. Jeliazkova, N., Kochev, N.: AMBIT-SMARTS: Efficient Searching of Chemical Structures and Fragments. Molecular Informatics **30**(8), 707–720 (2011)
13. Kochev, N.T., Paskaleva, V.H., Jeliazkova, N.: Ambit-tautomer: An open source tool for tautomer generation. Molecular Informatics **32**(5-6), 481–504 (2013)
14. Rahman, S.A., Bashton, M., Holliday, G.L., Schrader, R., Thornton, J.M.: Small molecule subgraph detector (SMSD) toolkit. Journal of Cheminformatics **1**(1), 12 (2009)
15. Rahman, S.A., Cuesta, S.M., Furnham, N., Holliday, G.L., Thornton, J.M.: EC-BLAST: a tool to automatically search and compare enzyme reactions. Nat Meth **11**(2), 171–174 (2014)
16. Spjuth, O., Helmus, T., Willighagen, E.L., Kuhn, S., Eklund, M., Wagener, J., Murray-Rust, P., Steinbeck, C., Wikberg, J.E.: Bioclipse: an open source workbench for chemo-and bioinformatics. BMC Bioinformatics **8**(1), 59 (2007)
17. Spjuth, O., Alvarsson, J., Berg, A., Eklund, M., Kuhn, S., Mäsak, C., Torrance, G., Wagener, J., Willighagen, E.L., Steinbeck, C., *et al.*: Bioclipse 2: A scriptable integration platform for the life sciences. BMC Bioinformatics **10**(1), 397 (2009)
18. Jeliazkova, N., Jeliazkov, V.: AMBIT RESTful web services: an implementation of the OpenTox application programming interface. Journal of Cheminformatics **3**(1), 1–18 (2011)
19. Marth, C.J., Gallego, G.M., Lee, J.C., Lebold, T.P., Kulyk, S., Kou, K.G.M., Qin, J., Lilien, R., Sarpong, R.: Network-analysis-guided synthesis of weisaconitine d and liljestrandinine. Nature **528**(7583), 493–498 (2015)
20. Clark, A., Sarker, M., Ekins, S.: New target prediction and visualization tools incorporating open source molecular fingerprints for tb mobile 2.0. Journal of Cheminformatics **6**(1), 38 (2014). doi:10.1186/s13321-014-0038-2
21. Cannon, E., Mitchell, J.B.O.: Classifying the world Anti-Doping agency's 2005 prohibited list using the chemistry development kit fingerprint. In: Berthold, Glen, R., Fischer, I. (eds.) Computational Life Sciences II. Lecture Notes in Computer Science, vol. 4216, pp. 173–182. Springer, ??? (2006)
22. Spjuth, O., Berg, A., Adams, S., Willighagen, E.L.: Applications of the InChI in cheminformatics with the CDK and bioclipse. Journal of Cheminformatics **5**(1), 14 (2013)
23. May, J.W., Steinbeck, C.: Efficient ring perception for the Chemistry Development Kit. Journal of Cheminformatics **6**(1), 3 (2014)
24. Faulon, J.-L., Visco, J. Donald P., Pophale, R.S.: The signature molecular descriptor. 1. using extended valence sequences in qsar and qspr studies. Journal of Chemical Information and Computer Sciences **43**(3), 707–720 (2003)
25. Faulon, J.-L., Collins, M.J., Carr, R.D.: The signature molecular descriptor. 4. canonizing molecules using extended valence sequences. Journal of Chemical Information and Computer Sciences **44**(2), 427–436 (2004)
26. Rojas-Chertó, M., Kasper, P.T., Willighagen, E.L., Vreeken, R.J., Hankemeier, T., Reijmers, T.H.: Elemental composition determination based on MSn. Bioinformatics **27**(17), 2376–2383 (2011)
27. May, J.W.: Beam. GitHub (2013). https://github.com/johnmay/beam
28. Figueras, J.: Ring perception using Breadth-First search. J. Chem. Inf. Comput. Sci. **36**(5), 986–991 (1996). doi:10.1021/ci960013p
29. Berger, F., Flamm, C., Gleiss, P.M., Leydold, J., Stadler, P.F.: Counterexamples in chemical ring perception. JOURNAL OF CHEMICAL INFORMATION AND COMPUTER SCIENCES **44**(2) (2004). doi:10.1021/ci030405d
30. AUTHORS (2015). https://github.com/cdk/cdk/blob/master/AUTHORS
31. Murray-Rust, P., Rzepa, H.S.: CML: Evolution and design. Journal of Cheminformatics **3**(1), 44 (2011)
32. Hicklin, J., Moler, C., Webb, P., Boisvert, R.F., Miller, B., Pozo, R., Remington, K.: JAMA: A Java Matrix

Package (2012)
33. Rogers, D., Hahn, M.: Extended-connectivity fingerprints. Journal of Chemical Information and Modeling **50**(5), 742–754 (2010)
34. Hall, L.H., Kier, L.B.: Electrotopological State Indices for Atom Types: A Novel Combination of Electronic, Topological, and Valence State Information. Journal of Chemical Information and Modeling **35**, 1039–1045 (1995)
35. Klekota, J., Roth, F.P.: Chemical substructures that enrich for biological activity. Bioinformatics (Oxford, England) **24**(21), 2518–25 (2008)
36. Vidal, D., Thormann, M., Pons, M.: Lingo, an efficient holographic text based method to calculate biophysical properties and intermolecular similarities. Journal of Chemical Information and Modeling **45**(2), 386–393 (2005)
37. PubChem Substructure Fingerprint V1.3. ftp://ftp.ncbi.nlm.nih.gov/pubchem/specifications/pubchem_fingerprints.txt
38. Alvarsson, J., Eklund, M., Engkvist, O., Spjuth, O., Carlsson, L., Wikberg, J.E.S., Noeske, T.: Ligand-based target prediction with signature fingerprints. Journal of Chemical Information and Modeling **54**(10), 2647–2653 (2014)

Figures

Figure 1 - Atom type information specified for a sp3-hybridized carbon.

```
<at:AtomType rdf:ID="C.sp3">
  <at:categorizedAs rdf:resource="&cdkat;C.tetrahedral"/>
  <at:hasElement rdf:resource="&elem;C"/>
  <at:hybridization rdf:resource="&at;sp3"/>
  <at:formalCharge>0</at:formalCharge>
  <at:lonePairCount>0</at:lonePairCount>
  <at:formalBondType rdf:resource="&bo;single"/>
  <at:formalBondType rdf:resource="&bo;single"/>
  <at:formalBondType rdf:resource="&bo;single"/>
  <at:formalBondType rdf:resource="&bo;single"/>
</at:AtomType>
```

Figure 2 - Integrated example showing the rendering and SMILES parsing functionality.

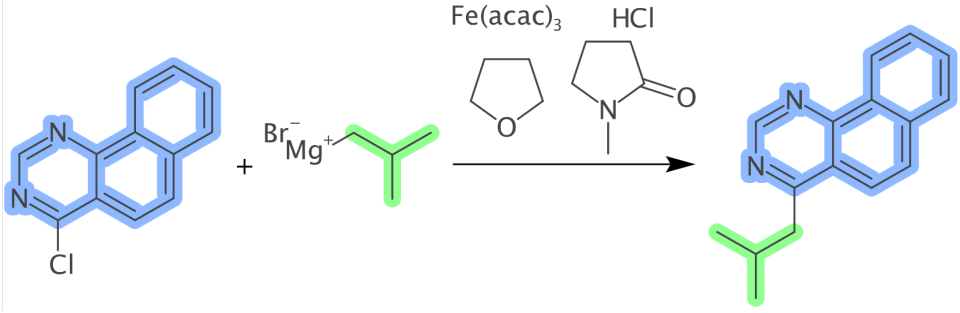


Figure 3 - The improved structure diagram generation has improved code to solve overlap. The original SDG code used general heuristics (left) and the OverlapResolver would fine tune the layout to ensure atoms would not be placed at the same location (middle). The new SDG algorithm is able to make more rigorous changes, making the final output must more pleasing (right).

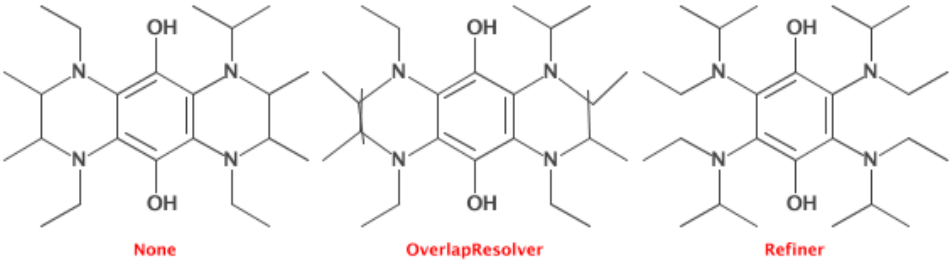
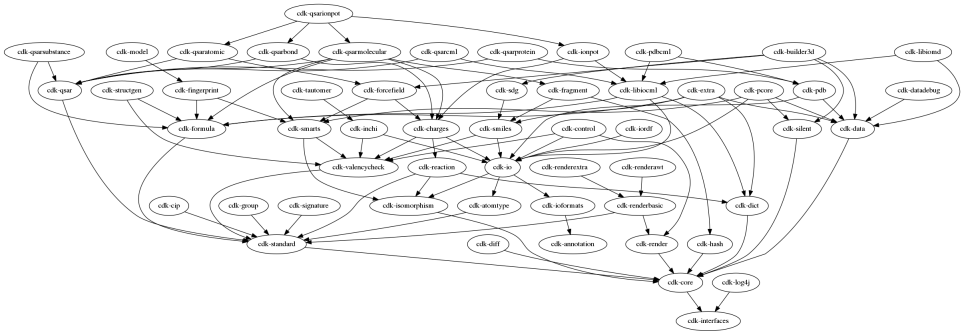


Figure 4 - Dependencies between CDK modules.
Visualization of the dependencies between CDK modules. For example, the cdk-core depends on the cdk-interfaces module. A few higher level modules have been left out: cdk-builder3dtools, cdk-legacy, and cdk-depict.



Tables

Table 1 - The molecular fingerprints in CDK.
Listed are the currently available molecular fingerprint in CDK with information about whether they come as bit / count versions, what CDK version they appeared in, their default size and where applicable also relevant references.

	Bit version	Count version	CDK version	Default Size
CircularFingerprinter [33, 20]	✓	✓	1.6.0 ?	$1\,024$ / 2^{32}[*]
EStateFingerprinter [34]	✓		1.2.0	79
ExtendedFingerprinter	✓		1.0	$1\,024$
Fingerprinter	✓		1.0	$1\,024$
GraphOnlyFingerprinter	✓		1.0	$1\,024$
HybridizationFingerprinter	✓		1.4.0	$1\,024$
KlekotaRothFingerprinter [35]	✓		1.4.6	$4\,860$
LingoFingerprinter [36]	✓		1.6.0 ?	NA[†]
MACCSFingerprinter	✓		1.2.0	166
PubchemFingerprinter [37]	✓		1.4.0	881
ShortestPathFingerprinter	✓		1.6.0 ?	$1\,024$
SignatureFingerprinter [38]	✓	✓	1.6.0 ?	2^{32}
SubstructureFingerprinter	✓		1.0	307

[*]For the CirkularFingerprinter the bit version is folded to 1024 whereas the count version is unfolded.
[†]The Lingofingerprinter does not have a default size.

Table 2 - A selection of key CDK modules with major changes.
An overview of a selection of often used CDK modules with description, dependencies on third-party libraries, and the major changes since version 1.2. Dependencies between modules are depicted in Figure 4.

Module	**Description**	**Major Changes**	**Dependencies**
interfaces			Vecmath 1.5.2
core			Google Guava 17.0
standard			
render		Redesigned to make it more modular and support multiple widget toolkits, like AWT and SWT.	
isomorphism			
atomtype		Unified approach where atom typing is separated from other algorithms.	
valencycheck			
ioformats			
io		The MDL molfile reader has been rewritten and supports in atom types defind in the specification.	XPP3 1.1.4c
iordf		New.	Jena 2.7.4
inchi			JNI-InChI 0.8 [22]
libiocml			XOM 1.2.5, CMLXOM 3.1 [31]
smiles		SMILES support performance and converage is greatly improved.	Beam 0.9.1 [27]
smarts			Beam 0.9.1 [27]
formula		New.	
fingerprint		Many new fingerprint types (see text).	Apache Commons Math 3.1.1
qsar			XOM 1.2.5, JAMA 1.0.3 [32]
signatures			Signatures 1.1
qsarmolecular			
log4j			Log4j 1.2.17