Chemistry Central Journal

ChemistryCentral

Software

Open Access

Cinfony – combining Open Source cheminformatics toolkits behind a common interface

Noel M O'Boyle*[1] and Geoffrey R Hutchison[2]

Address: [1]Cambridge Crystallographic Data Centre, 12 Union Road, Cambridge CB2 1EZ, UK and [2]Department of Chemistry, University of Pittsburgh, Chevron Science Center, 219 Parkman Avenue, Pittsburgh, PA 15260, USA

Email: Noel M O'Boyle* - oboyle@ccdc.cam.ac.uk; Geoffrey R Hutchison - geoffh@pitt.edu

* Corresponding author

Published: 3 December 2008

Chemistry Central Journal 2008, **2**:24 doi:10.1186/1752-153X-2-24

Received: 9 October 2008

Accepted: 3 December 2008

This article is available from: http://journal.chemistrycentral.com/content/2/1/24

Abstract

Background: Open Source cheminformatics toolkits such as OpenBabel, the CDK and the RDKit share the same core functionality but support different sets of file formats and forcefields, and calculate different fingerprints and descriptors. Despite their complementary features, using these toolkits in the same program is difficult as they are implemented in different languages (C++ versus Java), have different underlying chemical models and have different application programming interfaces (APIs).

Results: We describe Cinfony, a Python module that presents a common interface to all three of these toolkits, allowing the user to easily combine methods and results from any of the toolkits. In general, the run time of the Cinfony modules is almost as fast as accessing the underlying toolkits directly from C++ or Java, but Cinfony makes it much easier to carry out common tasks in cheminformatics such as reading file formats and calculating descriptors.

Conclusion: By providing a simplified interface and improving interoperability, Cinfony makes it easy to combine complementary features of OpenBabel, the CDK and the RDKit.

Background

Cheminformatics toolkits are essential to the day-to-day work of the practising cheminformatician. They enable the user to deal with such tasks as handling different chemistry file formats, substructure searching, calculation of molecular fingerprints, and structure diagram generation. The main Open Source cheminformatics libraries under active development are OpenBabel [1], the Chemistry Development Kit (CDK) [2], and the RDKit [3]. OpenBabel is a C++ toolkit with bindings in Perl, Python, Ruby and Java, the CDK is a Java toolkit, while the RDKit is another C++ toolkit with Python bindings. While the CDK has its origins in academia, both OpenBabel and the RDKit originated in companies (OpenEye and Rational Discovery, respectively) and have subsequently been developed by the community under Open Source licenses.

In general, all of these toolkits share the same core functionality although the implementation details and underlying chemical model may differ. However, as a result of their independent development and history, each has functionality specific to itself and each toolkit supports different sets of file formats and forcefields, and can calculate different molecular fingerprints and molecular descriptors (Table 1). Despite the diversity of these toolkits and the potential benefits in being able to access all of them at the same time, there has been little work on interoperability between them. This has resulted in a balkanization of this field such that users of one toolkit rarely use another toolkit.

One way to achieve interoperability of chemical toolkits is through the use of standard file formats for exchange of

Table 1: Some features of toolkits which are not shared by all three toolkits.

CDK
A large number of descriptors (some overlap with RDKit)
Pharmacophore searching (like RDKit*)
Calculation of maximum common substructure
2D structure layout (like RDKit) and depiction
MACCS keys (also RDKit) and E-State fingerprints
Integration with the R statistical programming environment
Support for mass-spectrometry analysis (representations for cleavage reactions, structure generation from formulae)
Fragmentation schemes (ring fragments, Murcko)
3D structure generation using a template and heuristics (like OpenBabel)
3D similarity using ultrafast shape descriptors
Gasteiger π charge calculation
OpenBabel
Not just focused on cheminformatics
Supports a very large number of chemical file formats including quantum mechanics file formats, molecular mechanics trajectories, 2D sketchers
3D structure generation using a template method (like CDK)
Included in all major Linux distributions
Bindings available from several scripting languages apart from Python, as well as the Java and .NET platforms
Conformation generation and searching
InChI (also CDK) and InChIKey generation
Support for crystallographic space groups
Several forcefield implementations: UFF (also RDKit), MMFF94, MMFF94s, Ghemical
Ability to add custom data types to atoms, bonds, residues, molecules
RDKit
A large number of descriptors (some overlap with CDK)
Fragmentation using RECAP rules
2D coordinate generation (like CDK) and depiction
3D coordinate generation using geometry embedding
Calculation of Cahn-Ingold-Prelog stereochemistry codes (R/S)
Pharmacophore searching (like CDK)
Calculation of shape similarity (based on volume overlap)
Chemical reaction handling and transforms
Atom pairs and topological torsions fingerprints
Feature maps and feature-map vectors
Machine-learning algorithms

* Where the term "like" is used, it indicates that the implementation details differ.

data. For example, the CML project has defined a standardised XML format for chemical data [4], with successive releases refining and extending the original standard. The OpenSMILES effort [5] has attempted to resolve ambiguities in the published SMILES definition [6] to create a standard. While these efforts deserve support, they face inevitable problems achieving consensus and they require changes to existing software to support the standard. The large number of chemical file formats supported by OpenBabel (currently over 80) illustrates both the potential of achieving a standard as well as the difficulties.

An alternative is interoperability at the API (application programming interface) level. This has the advantage that it does require any changes to existing software. However, there are at least three barriers to overcome: the need for a programming language that can access all the toolkits simultaneously, the difficulty of exchanging chemical models between different toolkits, and differences in the API for core cheminformatics tasks shared by the toolkits.

Here we describe Cinfony, a Python module that overcomes these barriers to provide interoperability at the API level. Cinfony allows access to OpenBabel, the CDK, and the RDKit through a common interface, and uses a simple yet robust method to pass chemical models between toolkits. Pybel, one of the components of Cinfony, has been described previously [7]. It provides access to OpenBabel from standard Python. In this work, we show that the API developed for Pybel may be considered a generic API for accessing any cheminformatics toolkit. We describe the design and implementation of the Cinfony API for OpenBabel, the RDKit and the CDK. Next, we show how Cinfony simplifies the process of accessing the toolkits and how it can be used in practice to combine the power of the three Open Source toolkits. Finally, we dis-

cuss performance and some results from comparisons of the toolkits.

Implementation

Common Application Programming Interface

Cinfony presents the same interface to three cheminformatics toolkits, OpenBabel, the CDK and the RDKit. These are available through three separate modules: *openbabel*, *cdk* and *rdkit*. The API is designed to make it easy to carry out many of the common tasks in cheminformatics, and covers the core functionality shared by all of the toolkits. Table 2 gives an overview of the API. The complete API is available here (see Additional file 1).

The main class containing chemical information is the Molecule class. Rather than create a new chemical model, the Molecule class is a light wrapper around the molecule object in the underlying library, for example, around OBMol in the case of OpenBabel. Attribute values such as the molecular weight are calculated dynamically by querying the underlying molecule. This ensures that if the underlying OBMol, for example, is altered, the attribute values returned will still be correct. The actual underlying object (an OpenBabel OBMol, a CDK Molecule, or an RDKit Mol) can be accessed directly at any point.

The Molecule class also contains several methods that act on molecules such as methods for calculating fingerprints, adding hydrogens, and calculating descriptor values. This makes it easy to access these methods, and also brings them to the attention of the user. In the underlying toolkit these methods may not be present as part of the molecule class, and in fact, they can be difficult to find in the toolkit's API. For example, the Cinfony method Molecule.addh() adds explicit hydrogens to the molecule. Although the OBMol of OpenBabel has a corresponding method, OBMol.AddHydrogens(), the RDKit uses a global method, AddHs(Mol), while the CDK requires the user to instantiate a HydrogenAdder object, which can then be used to add hydrogens.

The Molecule methods described in the original Pybel API [7] have been extended to handle hydrogen addition and removal, structure diagram generation, assignment of 3D geometry to 0D structures and geometry optimisation using forcefields. Both the CDK and the RDKit are capable of 2D coordinate generation and 2D depiction. However, since OpenBabel currently has neither of these capabilities, a fourth toolkit, OASA, is used by Pybel for this purpose. OASA is a lightweight cheminformatics toolkit implemented in Python [8].

A new development in the latest version of OpenBabel is 3D coordinate generation and geometry optimisation using one of a number of forcefields. Since these methods are also available in the RDKit, and are under development in the CDK, two additional methods have been added to the Cinfony Molecule: make3D(), for 3D coordinate generation, and localopt(), for geometry optimisation. Particularly in the case of OpenBabel, these new methods simplify the process of generating 3D coordinates. Compare a single call to make3D() in Cinfony with the following OpenBabel code:

```
structuregenerator = openbabel.OBOp.Find
Type('Gen3D')

structuregenerator.Do(mol)

mol.AddHydrogens()
```

Table 2: An overview of the Cinfony API.

Class name	Purpose
Molecule	Wraps a molecule instance of the underlying toolkit and provides access to methods that act on molecules
Atom	Wraps an atom instance of the underlying toolkit
MoleculeData	Provides dictionary-like access to the information contained in the tag fields in SDF and MOL2 files
Outputfile	Handles multimolecule output file formats
Smarts	Wraps the SMARTS functionality of the toolkit in an analogous way to the Python 're' module for regular expression matching
Fingerprint	Simplifies Tanimoto calculation of binary fingerprints
Function name	
readfile	Return an iterator over Molecules in a file
readstring	Return a Molecule
Variable name	
descs	A list of descriptor IDs
forcefields	A list of forcefield IDs
fps	A list of fingerprint IDs
informatsaa	A list of input format IDs
outformats	A list of output format IDs

```
ff = openbabel.OBForceField.FindType("MMFF94")

ff.Setup(mol)

ff.SteepestDescent(50)

ff.GetCoordinates(mol)
```

The Cinfony API is identical for all of the toolkits. However, the values returned by particular API calls are not necessarily standardised across toolkits. This Cinfony design decision is in agreement with the Principle of Least Surprise [9]; when the user accesses the underlying toolkit directly, they will get the same result as found when using Cinfony. This design decision places the responsibility on the user to become familiar with differences in how the toolkits behave. For example, all of the toolkits allow the calculation of path-based fingerprints. These encode all paths in the molecular graph up to a path length of *P* into a binary vector of length *V*, but the default values for *V* and *P* are different for each toolkit: 1024 and 7 for OpenBabel, 1024 and 8 for the CDK, and 2048 and 7 for RDKit. Although it is possible to alter these parameters for the CDK and the RDKit and so standardise *V* and *P* to 1024 and 7 for all of the toolkits, it is reasonable to assume that the developers of each package have chosen sensible defaults. In addition, the implementation details of each of the fingerprinters would still be different; for example, the RDKit sets four bits when hashing each molecular path, the others set one; OpenBabel does not set any bits for the one-atom fragments, N, C and O.

Interoperability

The ability to transfer chemical models between toolkits is essential to the goal of interoperability. However, the internal representation of a molecule is specific to a particular toolkit. For example, as well as the connection table and coordinates (if present), it may include derived data relating to aromaticity, the number of implicit hydrogens on an atom, or stereochemical configuration. Fortunately, the problem of transfer and storage of chemical information has already been solved by the development of molecular file formats, of which over 80 are now supported by OpenBabel. Specifically, the MDL MOL file format [10] and the SMILES format [5,6] are shared by all three toolkits, and are used by Cinfony to exchange information on molecules with 2D or 3D coordinates (MOL file format), and no coordinates (SMILES format), respectively.

By using existing file formats rather than trying to interconvert the internal models themselves, Cinfony takes advantage of the existing input/output code of each toolkit which is well-tested and mature. In addition, the translation process is transparent to the user. However, the user should be aware of known limitations of particular readers or writers. For example, the SMILES parser in CDK 1.0.3 ignores atom-based stereochemistry and thus that information is lost if a 0D *rdkit* or *obabel* Molecule with atom-based stereochemistry is converted to a *cdk* Molecule.

Cinfony Molecules are interconverted using the Molecule() constructor. For example, if obabelmol is an *obabel* Molecule, then the corresponding *rdkit* Molecule can be constructed using rdkit.Molecule(pybelmol). This mechanism can also be used to interface Cinfony to other cheminformatics toolkits. The only requirements are that the object passed to the Molecule() constructor needs to have a _cinfony attribute set to True, and an _exchange attribute containing a tuple (0, SMILES string) or (1, MOL file) depending on whether the molecule is 0D or not.

Implementation

The Python scripting language has two main implementations. The most widely used implementation is the original reference implementation of Python in C, referred to as CPython when necessary to distinguish it from other implementations. The next most widely used implementation is Jython, an implementation of Python in Java. Although most users of Python do so through CPython, Jython scripts have the advantage of being able to access Java libraries natively. They can also be compiled into Java classes to be used from Java programs. Jython scripts are also useful in contexts where Java is required but it is more convenient to work in Python; for example, to implement a Java web servlet or a node in a Java workflow environment such as KNIME [11].

As discussed earlier, one of the barriers to interoperability is the requirement for a programming language that can simultaneously access more than one of the toolkits. From CPython it is possible to use Cinfony modules to connect to OpenBabel (*pybel*), the CDK (*cdkjpype*) and the RDKit (*rdkit*). From Jython, there are modules for OpenBabel (*jybel*) and the CDK (*cdkjython*). Convenience modules *obabel* and *cdk* are provided that automatically import the appropriate OpenBabel or CDK module depending on the Python implementation. The relationship between these Cinfony modules and the underlying cheminformatics libraries is summarised in Figure 1.

pybel and jybel

OpenBabel provides SWIG [12] bindings for both CPython and Java (among other languages). *pybel* is a wrapper around the CPython bindings, and has previously been described in detail [7]. *jybel* is an implementation of the Cinfony API that allows the user to access OpenBabel from Jython using the Java bindings. Despite the fact that

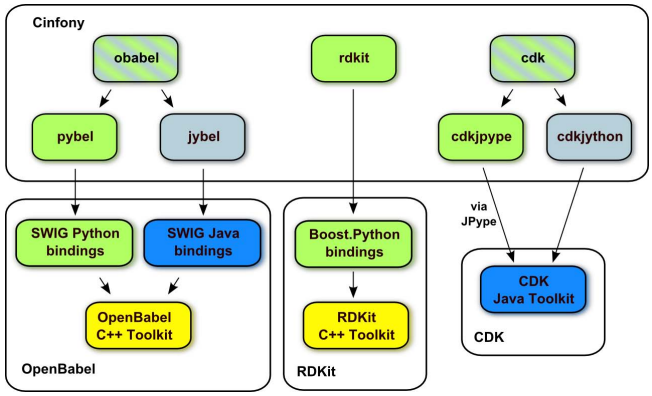

Figure 1
Relationship of Cinfony modules to Open Source toolkits. Python modules are accessible from CPython (green), Jython (pale blue), or both (striped green and pale blue). Java libraries are indicated by dark blue, while C++ libraries are yellow.

jybel is used from a Java implementation of Python, and accesses a C++ library through the Java Native Interface (JNI), the *jybel* code differs from *pybel* in very few respects. In Jython, it is not possible to iterate directly over the wrapped STL vectors used by OpenBabel as their Java SWIG bindings do not implement the Iterable interface. Also, the current Jython implementation is 2.2 and does not support generator expressions, which were introduced in Python 2.4. Although both C++ and Python have the concept of a global function or variable, this is not the case in Java. SWIG places such functions, and get/set methods for accessing the variables, in a special class named openbabel. Global constants are placed in another class called openbabelConstants. A convenience module, *obabel*, is provided which automatically imports the appropriate module depending on the Python implementation.

cdkjpype and cdkjython

Since Jython runs on top of the Java Virtual Machine (JVM), it can access Java libraries such as the CDK natively. To access Java libraries from CPython, the Python library JPype [13] is needed. This starts an instance of the JVM and uses the JNI to communicate back and forth. Overall, the differences between the two wrappers are minor. Jython and JPype differ in the syntax used to handle Java exceptions. Also, JPype returns unicode strings from the CDK and these need to be converted to regular strings (otherwise problems arise if they are passed to an OpenBabel method expecting a std::string). The appropriate CDK wrapper, *cdkjpype* or *cdkjython*, will be imported if the user imports the convenience module *cdk*.

rdkit

Support for Python scripting has been part of the design of the RDKit from the start. The Python bindings in RDKit were created using Boost.Python [14], a framework for interfacing Python and C++. The Cinfony module *rdkit* uses these bindings to implement its API. It is currently not possible to access RDKit from Jython. RDKit has only preliminary support for Java bindings; when these are complete, a corresponding module will be added to Cinfony.

Dependency handling

A fully-featured installation of Cinfony relies on a large number of open source libraries. In particular, the 2D depiction capabilities introduce dependencies on several graphics libraries which may be problematic to install on a particular platform (Cairo and its Python bindings, Python Imaging Library, AGG and the Python wrapper AggDraw). With this in mind, Cinfony treats all dependencies as optional and only raises an Exception if the user calls a method or imports a module that requires a missing dependency.

For example, the Python Imaging Library (PIL) is required for displaying a 2D depiction on the screen. If all of the components of cinfony are installed except for PIL, Cinfony works perfectly except that an Exception is raised if the Molecule.draw() method is called with show = True (the default). The image can however be written to a file without problems (show = False, filename = "image.png"). Similarly, if a user is only interested in using the CDK and the RDKit, it is not necessary to install OpenBabel.

Full installation instructions for Windows, MacOSX and Linux are available from the Cinfony website. It should be noted that for Windows users, there is no need to compile or search for missing libraries as the dependencies are included as binaries in the Cinfony distribution.

Results

Cinfony API

The original Pybel API was designed to make it easy to use OpenBabel to perform the most common tasks in cheminformatics and to do so using idiomatic Python. Subsequently, we realised that the resulting API could be considered a generic API for wrapping the core functionality of any cheminformatics toolkit. Cinfony implements an extended version of the original Pybel API for the CDK and the RDKit, as well as OpenBabel. While the original Pybel was restricted to CPython, Cinfony can also be used from Jython to access the CDK and OpenBabel.

Cinfony helps cheminformaticians avoid the steep learning curve associated with starting to use a new toolkit.

With Cinfony, all of the core functionality of the toolkits can be accessed with the same interface. For example, in Cinfony, a molecule can be created from a SMILES string with:

```
mol = toolkit.readstring("smi", SMILESstring)
```

RDKit

```
mol = Chem.MolFromSmiles(SMILESstring)
```

OpenBabel

```
mol = openbabel.OBMol()

obconversion = openbabel.OBConversion()

obconversion.SetInFormat("smi")

obconversion.ReadString(mol, SMILESstring)
```

CDK

```
builder = cdk.DefaultChemObjectBuilder.getInstance()

sp = cdk.smiles.SmilesParser(builder)

mol = sp.parseSmiles(SMILESstring)
```

The RDKit was designed with Python scripting in mind, and of the three toolkits is the most concise. On the other hand, OpenBabel uses a characteristically C++ approach. An empty molecule is created, and is passed to an OBConversion instance as a container for the molecule read from the SMILES string. The SmilesParser in the CDK requires an instance of an object implementing the IChemObjectBuilder interface.

Another advantage of a common API is that a script written for one toolkit can easily be modified to use another. As an example, here is a script that selects molecules that are similar to a particular target molecule. This script is taken from the original Pybel paper [7], but uses the CDK instead of OpenBabel and will run equally well from Jython and CPython. The only differences compared to the original script are that "pybel" has been replaced with "cdk", and the import statement has been changed from "import pybel":

```
from cinfony import cdk

targetmol = cdk.readfile("sdf", "targetmol.sdf").next()

targetfp = targetmol.calcfp()

output = cdk.Outputfile("sdf", "similarmols.sdf")

for mol in cdk.readfile("sdf", "inputfile.sdf"):

   fp = mol.calcfp()

   if fp | targetfp >= 0.7:

      output.write(mol)

output.close()
```

Alternatively, we could just have made a single change to the original script, by replacing the import statement from "import pybel" with "from cinfony import cdk as pybel".

Using Cinfony to combine toolkits

Another goal of Cinfony is to make it easy to combine toolkits in the same script. This allows the user to exploit the complementary capabilities of different toolkits (Table 1). For example, let's suppose the user wants to (1) convert a SMILES string to 3D coordinates with OpenBabel, then (2) create a 2D depiction of that molecule with the RDKit, next (3) calculate descriptors with the CDK, and finally (4) write out an SDF file containing the descriptor values and the 3D coordinates. The full Python script is only seven lines long:

```
from cinfony import rdkit, cdk, obabel

mol = obabel.readstring("smi", "CCC=O")

mol.make3D()

rdkit.Molecule(mol).draw(show = False, filename = "aldehyde.png")

descs = cdk.Molecule(mol).calcdesc()

mol.data.update(descs)

mol.write("sdf", filename = "aldehyde.sdf")
```

For cheminformaticians interested in developing QSAR or QSPR models, Cinfony can be used to simultaneously calculate descriptors from the RDKit, the CDK and OpenBabel. For example, the following script reads a multiline input file, with each line consisting of a SMILES string followed by a property value. For each molecule, it calculates all of the OpenBabel, RDKit and CDK descriptors (except for CDK's CPSA) and writes out the results as a tab-sepa-

rated file suitable for reading with the statistical package R [15]. Note that in this example script, if descriptors share the same name only one is retained. This is the case for the TPSA descriptor in OpenBabel, which is replaced by the RDKit's TPSA descriptor.

```
import string

from cinfony import obabel, cdk, rdkit

# Read in SMILES strings and observed prop
erty values

smiles, propvals = [], []

for line in open("data.txt"):

   broken = line.rstrip().split()

   smiles.append(broken [0])

   propvals.append(float(broken))

mols = [obabel.readstring("smi", smile)
for smile in smiles]

# Calculate descriptor values using
OpenBabel,

# the CDK (apart from 'CPSA') and the RDKit

cdkdescs = [x for x in cdk.descs if x !=
'CPSA']

descs = []

for mol in mols:

   d = mol.calcdesc()

   d.update(cdk.Molecule(mol).calcdesc(cd
kdescs))

   d.update(rdkit.Molecule(mol).calcdesc(
))

   descs.append(d)

# Write a file suitable for 'read.table'
in R

outputfile = open("inputforR.txt", "w")

descnames = sorted(descs [0].keys(), key =
string.lower)

print >> outputfile, "\t".join(["Prop
erty"] + descnames)

for smile, propval, desc in zip(smiles,
propvals, descs):

   descvals = [str(desc[descname]) for
descname in descnames]

   print >> outputfile, "\t".join([smile,
str(propval)] +

descvals)

outputfile.close()
```

Performance

Accessing cheminformatics libraries using Cinfony allows the user to rapidly develop scripts that manipulate chemical information. However, there is a small price to be paid. Firstly, there is the cost of moving objects across the interface between Python and the cheminformatics libraries. Secondly, the additional code required by Cinfony to implement a standard API may slow performance further.

To assess the performance penalty for accessing cheminformatics toolkits using Cinfony rather than directly in the native language, we looked at two simple test cases: (1) iterating over an SDF file containing 25419 molecules, (2) iterating and printing out the molecular weight of each of the molecules. The SDF file used was 3_p0.0.sdf, the first portion of the drug-like subset of the ZINC 7.00 dataset [16]. The Cinfony scripts, Java and C++ source code are available as Additional file 2. The results are shown in Table 3.

While accessing the CDK using Jython is almost as fast as a pure Java implementation, there is a considerable overhead associated with using JPype to access the CDK from CPython (89% slower for the second test case). This overhead is due to passing objects between the JVM and CPython. For OpenBabel, there is little performance cost associated with accessing OpenBabel from either implementation of Python, although the *jybel* scripts are somewhat slower than *pybel* scripts. A small portion of this speed difference can be attributed to a slower startup (about 1.6 seconds for *jybel*, compared to 0.8 seconds for *pybel*). Finally, from the RDKit results in Table 3, it is clear that using Boost.Python to wrap a C++ library is more efficient than using SWIG. The difference in run times between the C++ and Python implementations is negligible.

In practice, the performance of a particular Cinfony script will depend on the extent to which information is passed

Table 3: Performance of Cinfony modules compared to a native Java or C++ implementation.

	Iterate over SDF		Iterate and calculate molecular weight	
CDK	Time (s)	Normalised	Time (s)	Normalised
Native Java	21.2	1.00	36.8	1.00
cdkjython	23.1	1.09	41.6	1.13
cdkjpype	33.0	1.57	69.5	1.89
OpenBabel				
Native C++	31.9	1.00	43.0	1.00
pybel	34.1	1.07	45.1	1.05
jybel	38.0	1.19	49.6	1.15
RDKit				
Native C++	99.7	1.00	100.7	1.00
rdkit	99.9	1.00	101.0	1.00

The times reported are wallclock times from the best of three runs on a dual-core Intel Pentium 4 3.2 GHz machine with 1GB RAM.

back and forth between Python and the underlying Java or C++ library. Where most of the time is spent on computation in the underlying library, the speed difference between a native implementation and one using Cinfony is expected to be small.

Comparison of toolkits

Cinfony makes it easy to compare the results obtained by different toolkits for the same operations. This can be useful in identifying bugs, applying a test suite, or finding the strengths and weaknesses of particular implementations. For example, where different toolkits calculate the same descriptors, if the calculated values are not highly correlated it may indicate a bug in one or the other. Earlier, we mentioned that a difference in the treatment of implicit hydrogens causes different toolkits to give different values for molecular weight unless hydrogens are explicitly added. Ensuring that a particular result is in agreement with that obtained by another toolkit can act as a sanity check in such instances to avoid errors.

When carrying out the same operation with several toolkits, it is often convenient to iterate over the toolkits in an outer loop:

```
from cinfony import obabel, rdkit, cdk

for toolkit in [obabel, rdkit, cdk]:

   print toolkit.readstring("smi",
"CCC").molwt
```

As an example of how such comparisons can be used to identify bugs in toolkits, let us consider depiction. As a dataset, we randomly chose 100 molecules from PubChem [17], with subsequent filtering to remove multicomponent molecules. For each molecule, PubChem provides an SDF file containing coordinates for a 2D depiction, as well as the depiction itself as a PNG file. PubChem uses the CACTVS toolkit [18] to generate the 2D coordinates as well as the corresponding depiction. Using a script similar to the following, we used Cinfony to generate 2D depictions using OASA (the depiction library used by *pybel*), the CDK and a development version of RDKit that all use the same 2D coordinates taken from the SDF file:

```
from cinfony import pybel, rdkit

for toolkit in [rdkit, pybel]:

   name = toolkit.__name__

   for mol in toolkit.readfile("sdf",
"dataset.sdf"):

      mol.draw(filename = "%s_%s.png" %
(mol.title, name),

         show = False,

         usecoords = True)
```

When the resulting images were compared for the PubChem entry CID7250053, an error was found in the depiction of the stereochemistry of an isopropyl group (Figure 2). Since the error only occurred in certain cases, it had not been previously noticed and would have been difficult to identify without such a comparative study. Once reported, the problem was quickly solved and the subsequent RDKit release depicted the stereochemistry correctly. A comparison of depictions by commercial toolkits

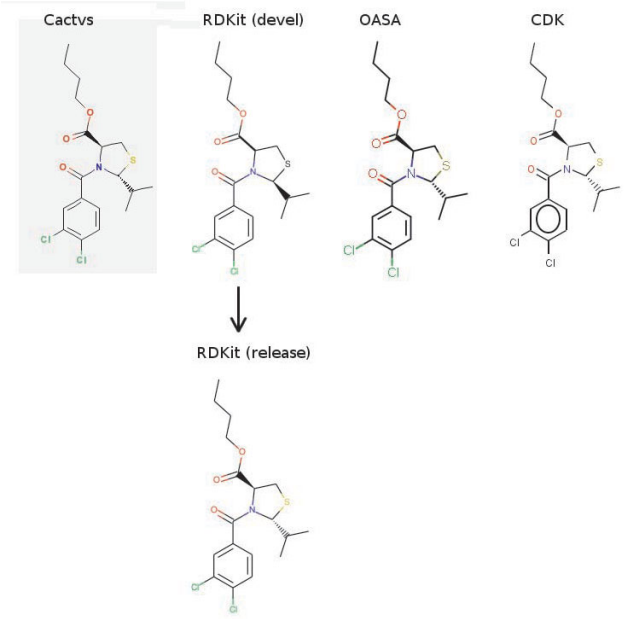

Figure 2
Comparison of depictions of PubChem CID7250053 using different toolkits. The depiction using the development version of RDKit showed incorrect stereochemistry for the isopropyl substituent of the thiazole ring.

and depictions generated by Cinfony is available here (see Additional file 3).

Conclusion

Cinfony makes it easy to combine complementary features of the three main Open Source cheminformatics toolkits. By presenting a standard simplified API, the learning curve associated with starting to use a new toolkit is greatly reduced, thus encouraging users of one toolkit to investigate the potential of others.

Cinfony is freely available from the Cinfony website [19], both as Python source code and as a Windows distribution containing dependencies. Installation instructions are provided for MacOSX, Linux and Windows.

Availability and requirements

Project name: Cinfony

Project home page: http://cinfony.googlecode.com

Operating system(s): Platform independent

Programming language: Python, Jython

Other requirements: OpenBabel, CDK, RDKit, Java, OASA, JPype, Python Imaging Library

License: BSD

Any restrictions to use by non-academics: None

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

NMOB conceived and developed Cinfony. GRH is the lead developer of OpenBabel and created the Python and Java SWIG bindings. All authors read and approved the final manuscript.

Additional material

Additional file 1

Miniwebsite API. *A mini-website of the Cinfony API documentation.*
Click here for file
[http://www.biomedcentral.com/content/supplementary/1752-153X-2-24-S1.zip]

Additional file 2

Timing Code. *A zip file containing Python, Java and C++ code used for run time comparisons for two test cases.*
Click here for file
[http://www.biomedcentral.com/content/supplementary/1752-153X-2-24-S2.zip]

Additional file 3

Miniwebsite Depictions. *A mini-website showing a comparison of the depictions generated by several cheminformatics toolkits.*
Click here for file
[http://www.biomedcentral.com/content/supplementary/1752-153X-2-24-S3.zip]

Acknowledgements

Cinfony would not be possible without the work of many Open Source projects. In particular, we thank several developers who responded quickly to bug reports or queries: Beda Kosata (OASA), Greg Landrum (RDKit), Tim Vandermeersch (OpenBabel), Steve Ménard (JPype). Thanks also to Gilbert Mueller and Chris Morley for feedback on installing Cinfony. NMOB thanks Google Code for providing free web hosting and development tools for Cinfony. We thank the anonymous reviewers for several useful suggestions.

References

1. **OpenBabel v.2.2.0** [http://openbabel.org]
2. Steinbeck C, Hoppe C, Kuhn S, Floris M, Guha R, Willighagen E: **Recent Developments of the Chemistry Development Kit (CDK) – An Open-Source Java Library for Chemo- and Bioinformatics.** *Curr Pharm Des* 2006, **12:**2110-2120.
3. Landrum G: **RDKit.** [http://www.rdkit.org].
4. Murray-Rust P, Rzepa HS: **Chemical Markup, XML, and the Worldwide Web. 1. Basic Principles.** *J Chem Inf Comput Sci* 1999, **39:**928-942.

5. Apodaca R, O'Boyle N, Dalke A, Van Drie J, Ertl P, Hutchison G, James CA, Landrum G, Morley C, Willighagen E, De Winter H: **OpenSMILES.** [http://www.opensmiles.org].
6. **Daylight Chemical Information Systems Manual** [http://www.daylight.com/dayhtml/doc/theory/theory.smiles.html]
7. O'Boyle NM, Morley C, Hutchison GR: **Pybel: a Python wrapper for the OpenBabel cheminformatics toolkit.** *Chem Cent J* 2008, **2:**5.
8. Kosata B: **OASA.** [http://bkchem.zirael.org/oasa_en.html].
9. Raymond ES: *The Art of UNIX Programming* 2003 [http://www.catb.org/~esr/writings/taoup/index.html]. Reading, MA: Addison-Wesley
10. **Symyx CTfile formats** [http://www.mdli.com/downloads/public/ctfile/ctfile.jsp]
11. **KNIME – Konstanz Information Miner** [http://knime.org]
12. **SWIG v.1.3.36** [http://www.swig.org]
13. Ménard S: **JPype.** [http://jpype.sf.net].
14. **Boost.Python** [http://www.boost.org/libs/python/doc/]
15. R development core team: **R: A language and environment for statistical computing.** [http://www.R-project.org].
16. Irwin JJ, Shoichet BK: **ZINC – A Free Database of Commercially Available Compounds for Virtual Screening.** *J Chem Inf Model* 2005, **45:**177-182.
17. **PubChem** [http://pubchem.ncbi.nlm.nih.gov/]
18. **CACTVS Chemoinformatics Toolkit: Xemistry GmbH: Lahntal, Germany.** .
19. O'Boyle NM: **Cinfony.** [http://cinfony.googlecode.com].