Rubabel: Wrapping OpenBabel with Ruby

Rob Smith,[1*]Ryan Williamson,[1] Dan Ventura[1] and John T Prince[2*]

[1]Department of Computer Science, Brigham Young University, Provo, UT, USA
[2]Department of Chemistry, Brigham Young University, Provo, UT, USA

Email: Rob Smith*- 2robsmith@gmail.com, John Prince*- jtprince@chem.byu.edu;

*Corresponding author

Abstract

Background The number and diversity of wrappers for chemoinformatic toolkits suggests the diverse needs of the chemoinformatic community. While existing chemoinformatics libraries provide a broad range of utilities, many chemoinformatists—particularly the programming-averse—find compiled language libraries intimidating, time-consuming, archane, and verbose. Although high-level language wrappers have been proposed, more can be done to leverage the intuitiveness of object-orientation, the concise paradigms of high-level languages, and the extensibility of languages like Ruby. We introduce Rubabel, an intuitive, object-oriented suite of functionality that substantially increases the accessibily of the tools in the OpenBabel chemoinformatics library.

Results Rubabel requires fewer lines of code than any other actively developed wrapper, providing better object organization and navigation, and more intuitive object behavior than extant solutions. Moreover, Rubabel provides a convenient interface to the many extensions currently available in Ruby, greatly streamlining otherwise onerous tasks such as creating web applications that serve up Rubabel functionality.

Conclusions Rubabel is powerful, intuitive, concise, freely available, cross-platform, and easy to install. We expect it to be a platform of choice for new users, Ruby users, and some users of current solutions.

Keywords: Chemoinformatics; OpenBabel; Ruby.

Background

Despite the fact that chemoinformatics tools have been proposed since the late 1990s [1], the field has yet to rally in support of a single library. The intricacies of the libraries combined with the low-level programming prowess required for these languages present a considerable barrier to adoption by less programming-oriented practitioners. What's more, the competing libraries don't share complete coverage of implemented tasks, meaning that the practitioner, who may be struggling with the language barrier, has to shoulder the additional burden of being well versed in the differences between the libraries, including different APIs, different IO interfaces and different data type standards.

The Cinfony project [2] is an attempt to offer high level access to three major existing chemoinformatics libraries from Python [3], a high-level scripting language [4]. Cinfony's use of Python greatly reduces the number of lines of code required for a broad range of chemoinformatics tasks. Though it allows the user to access the functionality of the component libraries from one Python script, Cinfony does not automatically manage underlying data types nor the choice of which library to use for which function. This allows users more control over how Cinfony works but, as the authors acknowledge, it requires users to have an intimate knowledge of the component libraries in order to manage what data types, conventions, and operations can be performed by each of the three libraries it wraps. Despite the success of Cinfony, there is still a need for simplified high level access to common chemoinformatics tasks.

Since most common tasks are available in any single chemoinformatics library, wrappers for single tool kits are widely used. Because these wrappers interface into a single library, they have the potential for simpler interfaces and easier extension.

Pybel [5, 6], a Python toolkit based on the Daylight project [7, 8], wraps the chemoinformatics library OpenBabel [9]. Pybel is an actively developed high-level solution to the accessibility problem of the available chemoinformatics libraries with an active user base. Still, Pybel's implementation in Python may not be the most intuitive interface for new users, who may not be strong programmers, or for Rubyists, who will miss multi-lined lambdas, simple extension (i.e, open-classes), and Rubygems, Ruby's streamlined add-on installation tool [10].

In addition to Pybel, other attempts have been made to make open source chemoinformatics libraries more accessible. Indigo Python, a Python wrapper bundled with the Indigo open source chemoinformatics library [11], is a substantial improvement over the C++ library it wraps in terms of reduction of lines of code (LOC) needed to implement common tasks. RDKit [12] is a C++ library that has a Python wrapper and provides substantial reduction of LOC over direct access to the underlying C++ library. Most other

available toolkits are either proprietary (such as OpenEye [13] and CACTVS [14]) or have not yet been documented and developed to maturity.

Ruby has penetrated the applied sciences where the need for a concise but powerful language meets appreciation for an easy learning curve [15, 16]. The attractiveness of a minimal learning curve, concise coding, and powerful language paradigm have made Ruby an attractive option for bioinformatics tools such as BioRuby [17]. In this paper we present Rubabel, the Ruby wrapper for the OpenBabel chemoinformatics toolkit.

For those who are not comfortable enough with programming to use the current tools, for those who prefer the Ruby way, and for those who want to do more with less lines of code, we present Rubabel. Rubabel offers a convenient, intuitive molecule-centric interface and facile intra-molecular navigation with minimal lines of code per task. It is an easily installed, actively developed project with a substantial amount of implemented functionality and an arbitrarily accessible extension mechanism for customization.

Implementation

Rubabel's architecture interfaces with OpenBabel through its Ruby SWIG bindings (see Figure 1). OpenBabel is an established chemoinformatics library written in C++ that provides a wide array of chemoinformatics functionality for programmatic or command line usage. OpenBabel supports 111 chemical file formats, including SMILES, SMARTS [18], and InChI. It has fingerprint support, bond perception, atom typing, image representation capabilities, stereochemistry recognition, and forcefields management, among other features. It's wide use is evidenced by the over 65 software applications, libraries, web applications, and databases that use it [11].

OpenBabel's acceptance rests at least partly on its SWIG bindings [19] which allow it to be accessed from languages other than C++. The bindings provide handles for accessing the internals of OpenBabel.

Ruby SWIG Bindings

For those who are less confident in C++ programming or aren't familiar enough with the code base to know the command line composition for their desired task, OpenBabel's Ruby SWIG bindings provide an alternative solution. Although the bindings technically allow access to OpenBabel from Ruby, it quickly becomes evident that the user is not convincingly spared from C++. An intimate understanding of OpenBabel's implementation architecture is required for many if not most tasks, and in some cases an almost line-for-line translation from C++ to Ruby is necessary. For example, Listing 1 shows how to instantiate a molecule

from a SMILES string with the Ruby bindings.

Listing 1: Creating a molecule from a SMILES string with OpenBabel Ruby bindings

```
obmol = OpenBabel::OBMol.new
obconv = OpenBabel::OBConversion.new
obconv.set_in_format("smi")
obconv.read_string(obmol, "CN2C(=O)N(C)C(=O)C1=C2N=CN1C")
```

Rubyists will notice that this code seems strikingly more like C++ than Ruby. Moreover, note that the despite the uncharacteristic simplicity of this example, the user still needs to understand explicit details of the OpenBabel architecture including the `OBMol` and `OBConversion` objects and modification methods for OBConversion. For more complex but typical examples, such as highlighting a substructure within a molecule in an image, the LOC required are comparable to C++. Adding Ruby-style objects and idioms to the SWIG bindings is the obvious next step toward improving upon the Ruby SWIG.

Rubabel is much more than a wrapper that ports OpenBabel functionality to Ruby. Rubabel organizes the OpenBabel objects into a more intuitive structure and extends the available behavior in a manner consistent with Ruby idioms, which is beneficial to experienced Rubyists and non-programmers alike, who will both find the interface intuitive and straightforward.

Rubabel: Augmentations to OpenBabel

Rubabel augments OpenBabel functionality with additional useful methods, as listed in Figure 2. Note that, because each Rubabel object fully encapsulates the corresponding underlying OpenBabel object, all native OpenBabel functions are accessible through the corresponding Rubabel object's OB member. Listed within each Rubabel object are a list of some of the novel methods implemented in Rubabel that are not available in OpenBabel. Most of these methods are not available in any other chemoinformatics toolkit.

Rubabel's objects are designed to be intuitive. Table 1 lists the Rubabel objects which wrap OpenBabel functionality. Although the names for these objects correspond to similarly named objects in OpenBabel, Rubabel augments OpenBabel functionality substantially. Figure 2 lists some of the novel methods offered by Rubabel. Additionally, every Rubabel object has full access to the behavior provided by the underlying OpenBabel object.

Rubabel: Ruby Idioms for Concise and Convenient Code

Ruby is a language designed to be easy to use, intuitive, brief, and fun. We designed Rubabel to embody as many of these admittedly subjective qualities as possible by designing Rubabel object behavior in ways

consistent with established Ruby idioms for object behavior.

OBJECT ORIENTATION Rubabel's object-oriented paradigm defines behaviors for the objects they interact with. For example, OpenBabel's Tanimoto coefficient logic will always apply to molecules, so in Rubabel that functionality is built into a method on the `Molecule` object. Similarly, in OpenBabel, write methods for drawing molecules in image files are located in the code base as stand-alone functions. However, object-oriented methodology dictates that the objects themselves—not external modules—should define how they are printed. In Rubabel, write methods are defined for `Molecule`. Another example of linking behavior to the objects modified can be found in Rubabel iterators. In OpenBabel, each object has it's own iterator type as a separate object. In Ruby, iterators are implicit and connected to the object that is iterated over. There is no need to look up behavior, because Rubabel's iterators work exactly like iterators over native Ruby objects.

By following the object-oriented paradigm users can instantly know what behavior is defined on any object by simply typing `<object name>.methods` in an interactive Ruby console. Non-object-oriented code requires digging through documentation or, if there isn't any, sourcecode. Both options are unappealing for the time commitment, while the latter option is inaccessible to non-programmers.

Listing 2 is illustrative of how object-orientation makes for more intuitive code. Through Rubabel's explicit `Bond` object, one can access a bond (line 3), upgrade it's order (line 3), and downgrade it's order (line 5). One would expect that the syntax to increase or decrease a bond's order would be by using + and -. With Rubabel, it is.

Listing 2: Ad-hoc bond modification

```
1 require "rubabel"
2 mol = Rubabel["CC"]
3 mol[0].get_bond(mol[1]) + 1   # now it is a double bond
4 bond = mol[0].bonds.first
5 bond - 1
6 bond.bond_order   # => 1
```

Object-orientation reduces lines of code. When considering an SD file, it seems reasonable to think about each entry in the file as a `Molecule` object. With Rubabel, you can do exactly that by iterating through each `Molecule` object in a file. Listing 3 shows how to open an SD file and print out each molecule whose weight is in the range (300,400) in just one line of code.

Listing 3: Report how many SD file records are within a certain molecular weight range

```
require 'rubabel'
puts Rubabel.foreach("benzodiazepine.sdf.gz").count {|mol| (300..400) ===
     mol.mol_wt }
```

STRING IDIOM To assist in convenience and minimize syntax lookup time, the Ruby idiom for strings is engineered for frequent exposure in Rubabel. For example, the `Molecule` object can be implicitly treated as a string, allowing splitting and matching operations that are concise and intuitive. For example, the user can automatically convert a text string into a molecule, then identify loose or exact matches, then optionally execute a block of code for each match, in this case adding a hydrogen atom (see Listing 4).

Listing 4: The concise power of the Ruby string idiom in Rubabel

```
require "rubabel"
Rubabel["C1CC12C3(C24CC4)CC3"].matches("*1**1"){|mol| mol.add_h!}
```

Additionally, Rubabel implements both split and append methods for the `Bond` object that mirror the same behavior defined in Ruby strings. Listing 5 shows an example of splitting bonds. Lines 2-4 create a molecule, find each single bond that links a carbon atom to an oxygen atom, then splits those bonds. Line 5 appends a carbon then an oxygen atom to `mol` using atomic numbers with the append function. Line 6 does the same using the element name.

Listing 5: Splitting and appending `Molecule` objects

```
require "rubabel"
mol = Rubabel["OCC"]
bonds = mol.matches("CO").map {|c, o| c.get_bond(o) }
mol.split(*bonds)
mol << 6 << 8
mol << :c << :o
```

Because molecules are treated as lists of atoms, you can quickly and easily access and modify specific atoms in a molecule. Listing 6 demonstrates adding an ethyl group to the first carbon atom by indexing into the `Molecule` (line 3).

Listing 6: Constructing a molecule atom-by-atom with the Ruby string idiom in Rubabel

```
require "rubabel"
mol = Rubabel["OCC"]
mol[1] << :c << :c
```

No other toolkits have equivalent functions to the string idiom in Rubabel.

ACCESS METHODS. Rubabel is designed to simplify common IO tasks to provide the shortest path to chemoinformatics functionality. Rubabel allows creation of `Molecule` objects from every format OpenBabel accepts, including SMILES strings (see Listing 7). Note that Rubabel requires only one line where the SWIG code requires 4 (compare with Listing 1).

Listing 7: Creating a molecule from a SMILES string with Rubabel

```
1 mol = Rubabel["CN2C(=O)N(C)C(=O)C1=C2N=CN1C"]
```

Efficiency in accessing objects is very important to reducing LOC and increasing intuition. Listing 8 gives some examples of object traversal in Rubabel, highlighting the amount of processing that can be done with very few lines of code in Rubabel. With very few lines of code and intuitive method names (select, find, reject), the user is able to conduct significant operations on newly created molecules. Lines 2-3 create a molecule then find the atom(s) that contain a double bond. Line 4 finds all the single- and double-bonded oxygen atoms in the molecule. Line 5 first finds all oxygen atoms, then removes from that list those that are bound to a carbon atom, yielding the peroxy oxygen.

Listing 8: Traversal of Objects in Rubabel

```
1 require "rubabel"
2 mol = Rubabel["NCC(O)CC(=O)CC"]
3 mol.find {|atom| atom.el == :o && atom.bonds.first.bond_order == 2 }
4 (two_bond_oxys, single_bond_oxys) = mol.select(&:oxygen?).partition(&:
      double_bond?)
5 mol.select {|atom| atom.el == :o }.reject {|atom| atom.any? {|at| at.el
      == :c}}
```

BUILDING. Rubabel offers multiple novel methods that assist in building and modifying molecules and bonds. Several, including the bond order increment/decrement operator, split, and match functions were already highlighted. Additionally, the `Molecule` object defines adding and removing atoms, as well as a `mass` method that calculates the mass of the molecule taking into account the charge state—a novel function not available in other toolkits.

BLOCKS. Blocks are dynamic sections of code with open scope, sometimes several lines long, that allow injection of specific behavior into otherwise generic methods. This allows greater code reuse, concise code, and places custom logic next to the object it modifies instead of in an external library. Consider Listing 9. By using `find` parameters in a block, Rubabel obtains a specific molecule in an SDF file in a more concise manner than RDKit, a Python chemoinformatics toolkit, which requires more control structure and logic.

Listing 9: Find a certain molecule in an SDF file

```
#Rubabel:
require "rubabel"
mol = Rubabel.foreach("benzodiazepine.sdf.gz").find {|mol| mol.title == "
    3016" }

#RDKit/Python
from rdkit import Chem
suppl = Chem.SDMolSupplier('benzodiazepine.sdf')
tgt=None
for mol in suppl:
    if not mol: continue
    if mol.GetProp('_Name')=='3016':
        tgt=mol
        break
```

Additionally, blocks make for easier synonymous code—code that is different syntactically but equivalent functionally. This increases the likelihood that a non-expert user can ascertain the syntax of desired operations with minimal reference to documentation while allowing more experienced users the freedom to use coding styles they are familiar and comfortable with.

Listing 10: Rubabel provides synonymous syntax

```
# find all alpha carbons
mol = Rubabel["NCCC(=O)CC(O)C=C"]
alpha_carbons = mol.select do |alpha_c|
  alpha_c.el == :c &&
  alpha_c.any? do |carbonyl_c|
    carbonyl_c.any? {|at| at.type == 'O2' }
  end
end

# another way to find all alpha carbons
alpha_carbons = mol.select do |alpha_c|
  alpha_c.el == :c &&
  alpha_c.any? do |carbonyl_c|
    carbonyl_c.any? do |at|
      at.el == :o &&
      at.bonds.all? {|bond| bond.bond_order == 2 }
    end
  end
end

# another way to find all alpha carbons
alpha_carbons = mol.select do |alpha_c|
  alpha_c.any? &:carbonyl_carbon?
end
```

In addition to the two examples given here, Listings 3 and 8 use blocks as well (lines 2 and 3-5, respectively). They are powerful tools not available in languages like C++ and Python.

CUSTOM BEHAVIOR. We have provided explicit `Molecule` methods for common tasks such as Tanimoto coefficient calculation, substructure highlighting, and graph diameter measurement. In the likely

event that users need custom extended behavior in Rubabel, they can take advantage of what are known as Ruby open classes. Objects in Ruby are more accessible to behavior modification than in some other languages. Writing custom behavior into Rubabel is analogous to using a plugin. Although OpenBabel has a plugin mechanism which allows external code to be integrated into the toolkit, it is not trivial to execute. In contrast, Rubabel can be modified and accessed with ease using Ruby's open classes. A class is open when it allows any external code to add or modify functionality in the local scope. For example, the Prince lab at BYU is currently developing a plugin for Rubabel that defines fragmentation behavior for lipid molecules. They require the molecule behavior defined by Rubabel and also need to add descriptions of how lipids fragment in order to accomplish their task. With Rubabel, this is as simple as adding a few lines in a new Ruby file, as in Listing 11.

Listing 11: Defining Custom Behavior for Rubabel. It is arbitrarily simple to add custom behavior to Rubabel by leveraging Ruby's open classes.

```ruby
require "rubabel"
class Molecule
  def new_behavior
    #add custom behavior here
  end
end
mol = Rubabel::Molecule.new
mol.new_behavior #use custom behavior here
```

By using Rubabel, custom behavior can be defined and shared amongst lab groups and colleagues rapidly and easily.

Rubabel: Extensions from Ruby

Rubabel has access to the Ruby community's many actively developed extensions (see Figure 1 and Table 2 for examples). These extensions and the many more like them provide diverse and useful benefits such as quicker programming, easy debugging, and easy installation. Some Ruby add-ons, like Sinatra [20], a concise web application framework, and Rspec [21], a test-driven development suite, have no equivalent that we are aware of in other languages such as Python.

Building a Rubabel Web App in Sinatra

As an example of the capabilities of these extensions, consider Sinatra. Using Sinatra, it is possible to give practitioners online access to Rubabel in very few lines of code. Applications could easily be developed to serve up the native functionality of Rubabel as well as custom functionality developed as needed. To

demonstrate the brevity of code required, consider the task of adding a hydrogen atom to a molecule and printing the SVG image of the new molecule. Assuming that the user has a standard install of Ruby, which includes Rubygems and the prerequisites for OpenBabel, the entire environment for Sinatra and Rubabel can be installed in two lines (see Listing 12).

Listing 12: Installing Rubabel and Sinatra.

```
gem install sinatra
gem install rubabel
```

The functionality for the web app requires only five lines of code (see Listing 13). We place these in the file `mol_h.rb`.

Listing 13: A web application that adds a hydrogen atom to a molecule.

```
require "sinatra"
require "rubabel"
get "/add_h/:mol" do |mol|
  Rubabel[mol].add_h!.write("test.svg")
end
```

Now, to invoke our web server locally, we simply open a terminal and write: `ruby mol_h.rb`

The web service is now available. Now we can convert a smiles string to a molecule, then add a hydrogen and print the resulting molecule simply by typing `http://0.0.0.0:4567/add_h/C` into a browser window. This results in a web page that displays the svg of the resulting molecule (see Figure 3). The address `http://0.0.0.0:4567/` accesses the local web server. The argument `add_h` tells Rubabel that we want to add a hydrogen onto the last argument of the url, the SMILES string `C`.

The simplicity of this example readily extends to all facets of Rubabel.

IRB

Though space will not permit an exhaustive consideration of all Ruby extensions that can be used in conjunction with Rubabel, the interactive Ruby shell (IRB) is of special import. As with languages like Python, Ruby's interactive shell allows users a ready sandbox to run quick experiments, test syntax, or debug their scripts. IRB can be installed (provided the user has Rubygems) by typing `gem install irb`. Simply enter the IRB environment (`irb` at the terminal) and type `require 'rubabel'` and all of Rubabel's functionality is accessible in an interactive terminal. This is particularly useful given the number of tasks that Rubabel can accomplish in just one line. Rubabel in IRB provides an interactive sandbox to experiment in realtime with instant feedback—a refreshing alternative to stringing together guess-and-check command line arguments.

As mentioned before, this is also a fantastic and fast way to look up (via `<object name>.methods` or check syntax.

Results and Discussion

To provide a quantitative analysis of Rubabel compared to existing tools, we use a lines of code (LOC) comparison from the Chemistry Toolkit Rosetta Wiki [22]. The CTRwiki provides code snippets for 18 common chemoinformatics tasks for more than 17 toolkits in various programming languages. Since there are several toolkits with only one or two solutions we consider only open source solutions with at least 5 of the CTR tasks implemented.

Rubabel dominates Indigo C++ in number of lines of code per task, and is more concise than other scripting language toolkits (see Figure 4). Rubabel has less lines of code per task on average than Pybel. Rubabel also implements almost double the CTR tasks of Pybel (see Figure 5), and when broken out by task, we can see that Rubabel is more concise than Pybel on each task for which they are both implemented save one (task 9) (see Figure 6). Moreover, Rubabel is more concise than all other methods for each task save rdkit/Python on task 10.

Rubabel has some features which users may find useful that are not available in Pybel. These include an explicit `Bond` object and the associated functionality, simpler atom interrogation, enumerable atoms and bonds, wrapped output and input options (obviating the need to dig through OpenBabel documentation to parse them out programmatically), more `Molecule` object modifications (e.g. adding hydrogens at a specific pH), and simpler output (Rubabel infers the output format from the filename). Additionally, there are several extensions written in Ruby that do not yet have equivalents in Python (see Table 2).

Rubabel is open source software released under the liberal MIT license. The license and source code, as well as instructions on how to install, are found at https://github.com/princelab/rubabel. The project is available as a Ruby gem [10], which makes it exceedingly easy to install. For those who already have Ruby, Rubygems, and OpenBabel's prerequisites installed, Rubabel and all requirements (including OpenBabel) can be installed with one line: `gem install rubabel`. Rubabel can also be downloaded and built from source. The instructions for this are available on the github site mentioned above.

Conclusions

Chemists are not necessarily computer scientists. The more concise, clear, and accessible a toolkit is, the less time they spend learning syntax and the more time they spend solving chemistry problems. Ruby

is designed to be intuitive, concise, and powerful. Rubabel wraps OpenBabel in a way that is true to these qualities. Rubabel provides more intuitive object organization than OpenBabel and provides extra functionality designed to streamline code writing by limiting the time necessary to look up function syntax and the number of lines of code required. Rubabel also provides access to the many open source extensions available for Ruby. Rubabel's concise and intuitive design makes common chemoinformatics tasks readily accessible from scripts, interactive shells, or custom applications in few lines of code and with less time spent learning APIs. Intentionally intuitive design, concise code idioms, and simplified common tasks make Rubabel appealing to Rubyists, non-programmers, and a segment of the users of other platforms.

Availability and Requirements

Project name: Rubabel

Project home page: https://github.com/princelab/rubabel

Operating System(s): Platform independent

Programming language: Ruby

Other requirements: OpenBabel's Install Requirements, Rubygems

License: MIT

Any restrictions to use by non-academics: None

Competing Interests

None.

Author's contributions

JP is the founding developer of Rubabel. RS and RW extended Rubabel. DV provided valuable guidance and editing. All authors read and approved the final manuscript.

Acknowledgements

RS acknowledges the NSF (DGE-0750759) for financial support.

References

1. Hann M, Green R: **Chemoinformatics — a new name for an old problem?** *Current Opinion in Chemical Biology* 1999, **3**(4):379–383, [http://www.sciencedirect.com/science/article/pii/S136759319980057X].
2. O'Boyle NM, Hutchison GR: **Cinfony – combining Open Source cheminformatics toolkits behind a common interface**. *Chemistry Central journal* 2008, **2**:24, [http://dx.doi.org/10.1186/1752-153X-2-24].
3. **Python**. *http://www.python.org*.
4. Ousterhout J: **Scripting: Higher Level Programming for the 21st Century**. *http://www.home.pacbell.net/ouster/scripting.html*.
5. O'Boyle N, Morley C, Hutchison G: **Pybel: a Python wrapper for the OpenBabel cheminformatics toolkit**. *Chemistry Central Journal* 2008, **2**:1–5, [http://dx.doi.org/10.1186/1752-153X-2-5]. [10.1186/1752-153X-2-5].
6. **OpenBabel Python**. *http://openbabel.sourceforge.net/wiki/Python*.
7. **Daylight Toolkit: Daylight Chemical Information Systems, Inc.: Aliso Viejo, CA**.
8. **PyDaylight: Dalke Scientific Software, LLD: Santa Fe, NM**.
9. O'Boyle N, Banck M, James C, Morley C, Vandermeersch T, Hutchison G: **Open Babel: An open chemical toolbox**. *Journal of Cheminformatics* 2011, **3**:1–14, [http://dx.doi.org/10.1186/1758-2946-3-33]. [10.1186/1758-2946-3-33].
10. **Rubygems**. *http://www.rubygems.org*.
11. Pavlov D, Rybalkin M, Karulin B, Kozhevnikov M, Savelyev A, Churinov A: **Indigo: universal cheminformatics API**. *Journal of Cheminformatics* 2011, **3**:1–1, [http://dx.doi.org/10.1186/1758-2946-3-S1-P4]. [10.1186/1758-2946-3-S1-P4].
12. **RDKit**. *http://www.rdkit.org*.
13. **OEChem: OpenEye Scientific Software: Sante Fe, NM**. *http://www.eyesopen.com*.
14. Ihlenfeldt WD, Takahashi Y, Abe H, Sasaki S: **Computation and management of chemical properties in CACTVS: An extensible networked approach toward modularity and compatibility**. *Journal of Chemical Information and Computer Sciences* 1994, **34**:109–116, [http://pubs.acs.org/doi/abs/10.1021/ci00017a013].
15. **Ruby**. *http://www.ruby-lang.org/*.
16. **SciRuby**. *http://www.sciruby.com*.
17. Goto N, Prins P, Nakao M, Bonnal R, Aerts J, Katayama T: **BioRuby: bioinformatics software for the Ruby programming language**. *Bioinformatics* 2010, **26**(20):2617–2619, [http://bioinformatics.oxfordjournals.org/content/26/20/2617.abstract].
18. **SMARTS - A Language for Describing Molecular Patterns**. *http://www.daylight.com/dayhtml/doc/theory/theory.smarts.html*.
19. **SWIG**. *http://www.swig.org*.
20. **Sinatra Ruby Web Framework**. *http://www.sinatrarb.com*.
21. **RSpec**. *http://rspec.info/*.
22. **Chemistry Toolkit Rosetta Wiki**. *http://ctr.wikia.com/wiki/*.
23. **Rubyvis, a Ruby Graphical Plotting Library**. *http://rubyvis.rubyforge.org/*.
24. **Ruby-debug**. *http://bashdb.sourceforge.net/ruby-debug.html/*.

Figures

Figure 1- Rubabel Architecture

Rubabel reorganizes OpenBabel functionality in an object-oriented architecture via the Ruby SWIG bindings and adds significant novel functionality. Additionally, Rubabel opens the possibility of integrating Ruby's

substantial library of extensions, providing debugging tools (Ruby Debugger), code testing (Rspec), graphic visualizations (Rubyvis), rapid dissemination of tools (Rubygems), web interfaces (Sinatra), and scientific libraries (Sciruby).

Figure 2- Novel Functionality in Rubabel

Besides providing access to native OpenBabel functions, Rubabel provides a host of novel functionality.

Figure 3 - Custom Web Apps with Rubabel

The Sinatra toolkit for Ruby allows easy web access for Rubabel and add-ons.

Figure 4 - Average Lines of Code per CTR Task

On average, Rubabel requires fewer lines of code than any other toolkit.

Figure 5 - Number of CTR Tasks Implemented

Rubabel has more tasks implemented than any other toolkit.

Figure 6 - Lines of Code Per CTR Task

Rubabel has more CTR tasks implemented than any other toolkit, and also has less lines of code than any other toolkit on every task except task 9, where Pybel has one less line of code, and task 10, where rdkit/Python is slightly more concise.

Tables

Table 1 - Rubabel Objects

Rubabel's object organization is an intuitive restructuring of OpenBabel's architecture. For example, molecule printing logic found in an external object in OpenBabel are moved inside the Molecule object. Rubabel's objects have extended novel capabilities (detailed below).

Molecule	Wraps OpenBabel's OBmol object. Adds the ability to intelligently manipulate molecules as strings, transfer to and from lists of atoms and bonds, add and modify atoms, explicit and general molecular matching, iterate over bonds or atoms, copy molecules, png representation of the molecule, and fingerprinting.
Atom	Wraps OpenBabel's OBatom object. Adds accessibility conveniences such as the ability to seamlessly create or access an atom as an atomic number, the ability to intrinsically iterate through bonds and pass blocks to iterating loops, and the ability to iterate through and optionally execute a block of code for each atom bonded to the current one.
Bond	Wraps OpenBabel's OBBond object. Adds an accessor for a list of attached atoms, a seamless enumerator for attached atoms, the ability to execute a block of code for each attached atom, and the ability to easily check if a given atom is connected with this bond.
Smarts	Wraps OpenBabel's smarts pattern object.

Table 2 - Ruby Extensions Accessible to Rubabel

Ruby has an active community of contributors who are constantly developing open source tools and frameworks.

Extension	**Possible Application with Rubabel**
Sinatra [20], a web application framework	Quick and easy webapp GUI for OpenBabel, allowing multi-platform point and click chemoinformatics
Sciruby [16], a scientific library	Plotting, statistical tools, access to R programming language for Rubabel results
Rubyvis graphical library [23]	Open ended graphical software to make clean representations of numerical data
IRB, the interactive Ruby shell	Quick access to Rubabel and OpenBabel from a terminal
Rspec, an automated code testing library [21]	Automated unit tests for software built with Rubabel (No Python equivalent due to Ruby's block ability)
Ruby debugger [24]	Step into executed code with a live IRB session to ferret out bugs
Rubygems, a distribution tool [10]	Easily distribute and integrate applications written with Rubabel with a one-line install