The Cactvs Cheminformatics Toolkit in the Python Age

Xemistry chemoinformatics

Wolf-D. Ihlenfeldt, Königstein, Germany

The Cactvs Toolkit and the Interface Language Conundrum

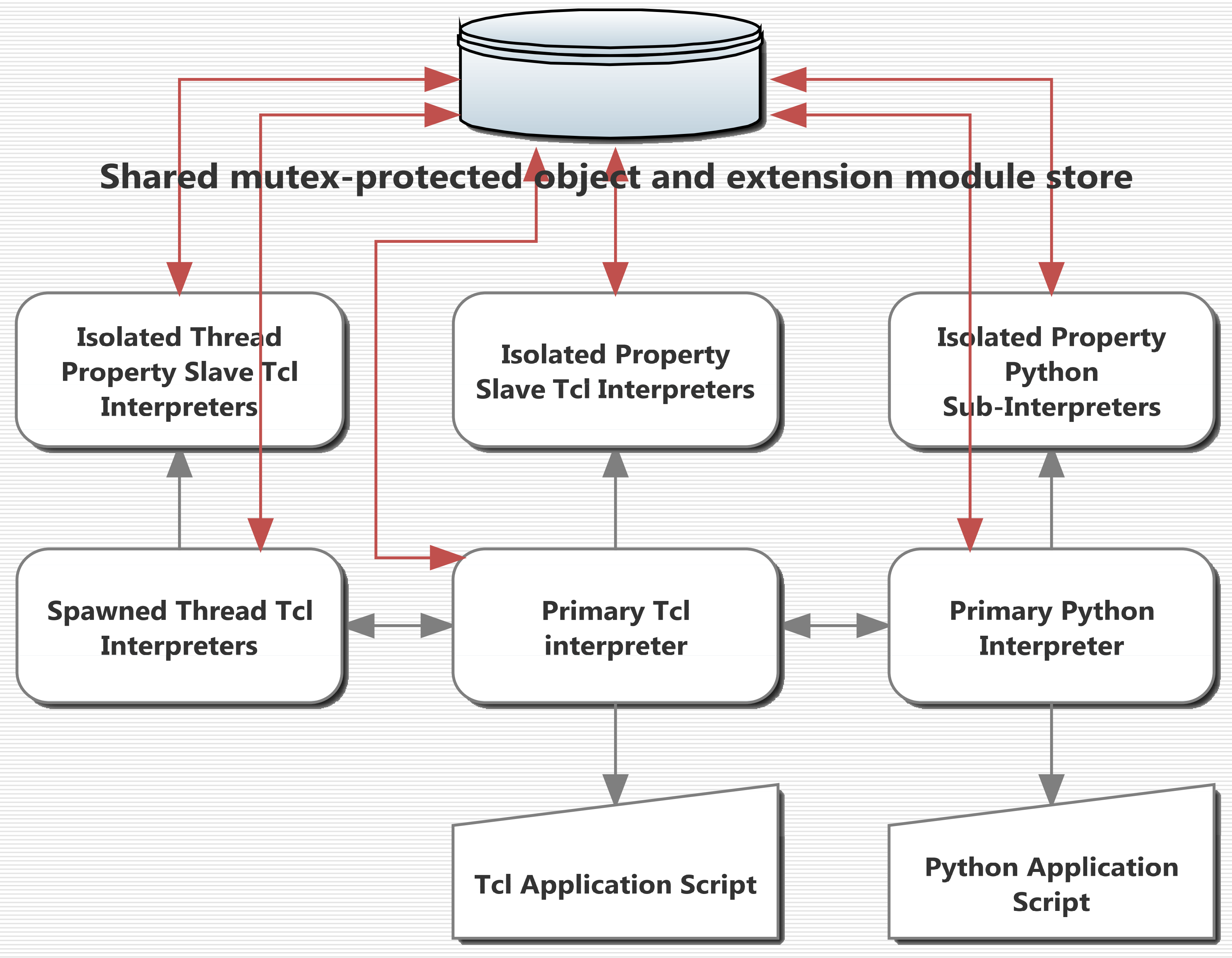
Cactvs is a general-purpose chemical information processing toolkit with an extensive set of features. It processes structure, reaction, table and network data. It can read and write over a hundred structure, reaction, table and network data formats, talk to many databases, and has extensive support for Internet-based information resources and standards. It also computes a comprehensive suite of structure and reaction property data – and does this all in distributed or multi-threaded fashion if desired. It is still the only toolkit which solves all (pretty simple) test problems posted on the Chemistry Toolkit Rosetta Wiki (http://ctr.wikia.com/wiki/Chemistry_Toolkit_Rosetta_Wiki).

The toolkit is designed for rapid solution development by scripting. Its original interface language is Tcl - a capable but not very well known and slightly dated language which nevertheless provides multi-threading features and multi-interpreter insulation far beyond what is possible in, for example, Python.

Still, Python has definitely become the accepted standard for cheminformatics and bioinformatics data processing. Most competing toolkits have a Python interface, or are even loadable as a module into a generic Python interpreter.

For these reasons, providing a Python interface to the Cactvs toolkit has become essential. This has now been implemented.

Quite a number of components of the toolkit are itself scripted in Tcl – for example encapsulated standard property computation modules. Legacy components need to remain usable – whether a module is implemented in Tcl or Python must be irrelevant regardless in which language the application script is written. This leads to a unique multi-language approach – the toolkit now interfaces to both languages simultaneously, in a common executable, with bridges between them.

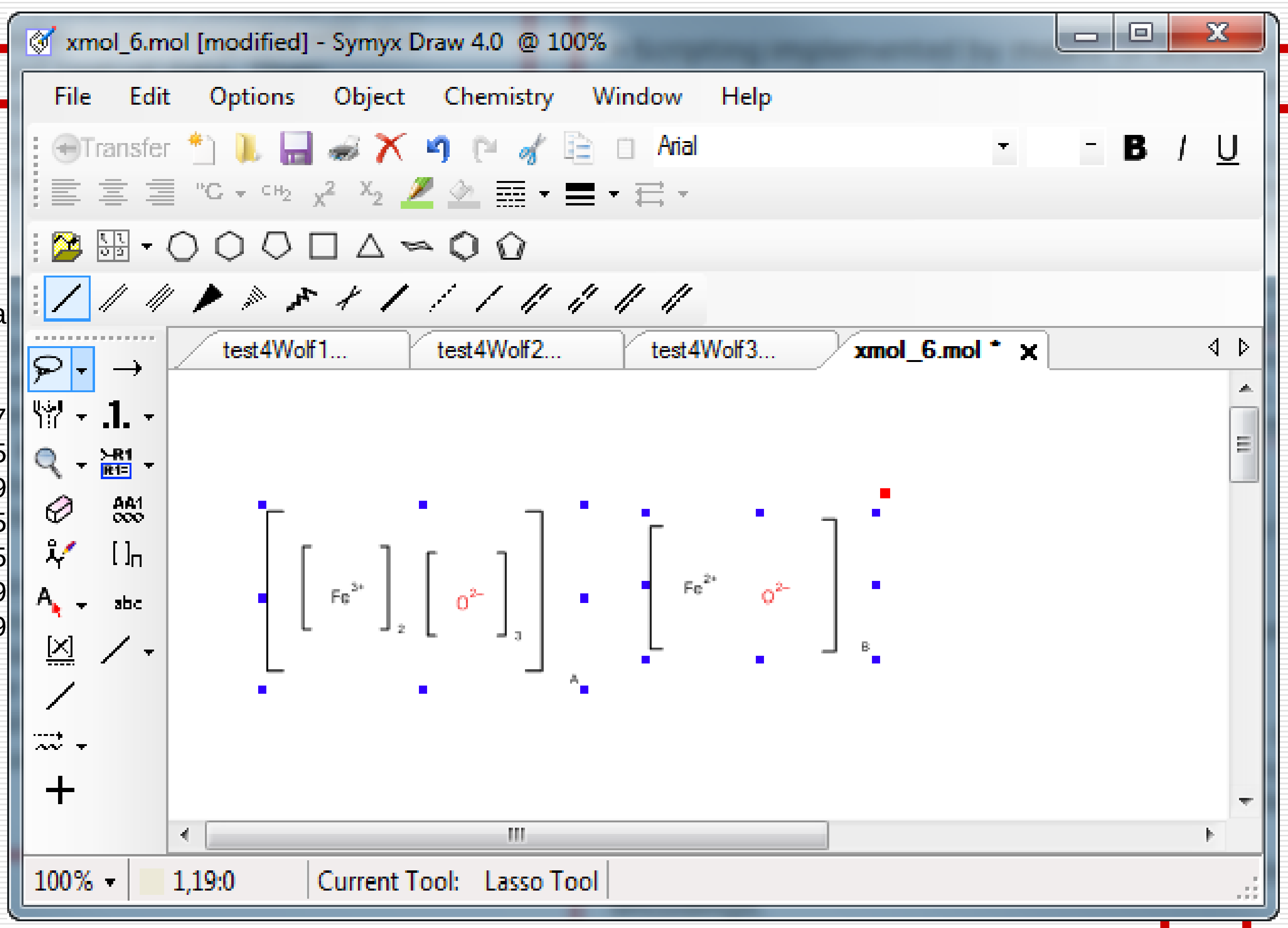

Shared mutex-protected object and extension module store
Isolated Thread Property Slave Tcl Interpreters
Isolated Property Slave Tcl Interpreters
Isolated Property Python Sub-Interpreters
Spawned Thread Tcl Interpreters
Primary Tcl interpreter
Primary Python Interpreter
Tcl Application Script
Python Application Script

Design Complications

Objects (structure, reactions, tables, networks, files, datasets) are shared between interpreters by default. Any change performed in one language environment is automatically mirrored in the other.

One primary interpreter and potentially multiple additional Tcl thread interpreters may operate on these objects simultaneously.

The active primary interpreter may invoke its counterpart of the other language for specific tasks. It is even possible to have application scripts consisting of a Tcl and a Python part, for example for re-use of in-house procedure libraries.



```
ACCLDra

  7  0
  12.47
  11.25
   7.79
   5.85
   5.85
   7.79
   7.79
M  CHG
M  STY
M  SLB
M  SPL
M  SAL
M  SDI
M  SDI
M  SDT
M  SDD   2      9.3481   -6.1856    DA    ALL  1       5
M  SED   2 A
M  SAL   3  2   2   1
M  SDI   3  4   10.6681   -5.7256   10.6681   -3.8456
M  SDI   3  4   13.5081   -3.7356   13.5081   -5.7656
M  SDT   4 FDAREG_SGROUP                F
M  SDD   4     13.8081   -5.6856    DA    ALL  1       5
M  SED   4 B
M  SAL   5  3   3   6   7
M  SPA   5  1   3
M  SDI   5  4    7.2581   -5.5156    7.2581   -4.2656
M  SDI   5  4    8.4481   -4.2656    8.4481   -5.5156
M  SMT   5 3
M  SAL   6  2   4   5
M  SPA   6  1   4
M  SDI   6  4    5.3481   -5.4156    5.3481   -4.1656
M  SDI   6  4    6.6581   -4.1656    6.6581   -5.4156
M  SMT   6 2
M  END
```

Example Mini Project in Tcl and Python

Does your toolkit of choice have a *"Molfile Reader"* – or a **MOLFILE READER**?

The task: Process a set of files with multiple nested, overlapping SGroups of various types, with incomplete atom sets, superatoms and badly aligned annotations. Split the files into individual components by *datagroup* annotation labels and write a multi-record SDF, preserving the proper subset of SGroups, annotations and all the other M xxx stuff in the output. It seems the FDA needs to do this regularly.

In both language variants, this is a simple exercise for the toolkit:

```
filter create datagroup property G_TYPE value datagroup operator =
set fhout [molfile open "split.sdf"  w]
set eh [molfile read "xmol_6.mol"]
foreach g [ens groups $eh datagroup] {
   set ehg [group dup $eh $g]
   molfile write $fhout $ehg
   ens delete $ehg
}
molfile close $fhout
ens delete $eh

f=Filter('datagroup',property='G_TYPE',value='datagroup',operator='=')
with Molfile('split.sdf','w') as fhout:
   with Molfile.Read('xmol_6.mol') as eh:
      for g in eh.groups(f):
        with g.dup() as ehg:
           fhout.write(ehg)
```

Availability

V3.426, which includes Python interfaces to almost all the objects and functions of the Tcl toolkit (exception: concurrent multi-interpreter multi-threaded script execution - generally unsupported in Python) is available for public download from http://www.xemistry.com/academic.

FAQ

- *Python 2 or 3?* 3.4.1. The interface relies on new features only introduced in 3.4.

- *Where do I get this brand-new Python version?* No need. The toolkit includes a complete Python installation. It coexists with other installed Python versions.

- *Can I load other Python-linked chemistry toolkits into this system?* Yes, in principle. But remember, its Python3. You cannot load Python2 compiled modules.

- *Is it Open Source?* No. It is free software for academia, though. You may freely redistribute anything you develop for it. Scripting and module APIs are public and royalty-free.

Supporters

We gratefully acknowledge financial support by Vertex Pharmaceuticals for this project. All improvements implemented with their aid are available in the free academic releases.